NetOI Quick Start Guide

Version 2.0

REVELATION
SOFTWARE
A Division of Revelation Technologies, Inc.

Part No. 59-931

Printed in the United States of America.

Table of Contents

Section I: Developing .NET Applications for OpenInsight

Using the NetOI .NET assembly, you can code entirely in Visual Studio (or your development environment of choice), and develop a Windows application that uses OpenInsight as its data source. The NETOI assembly contains classes to communicate with the OpenInsight Engine Server (the Server class); open and communicate with files (the OIFile class); and manage records and selectlists (the OIRecord and OISelectList classes). Using NETOI, you can perform all the basic file I/O needed to use OpenInsight. Figure 1 is an example of some Visual Basic.NET code using these classes. In addition, the Server class allows you to invoke functions and subroutines on the host, as well as execute (and capture the results from) OpenInsight commands.

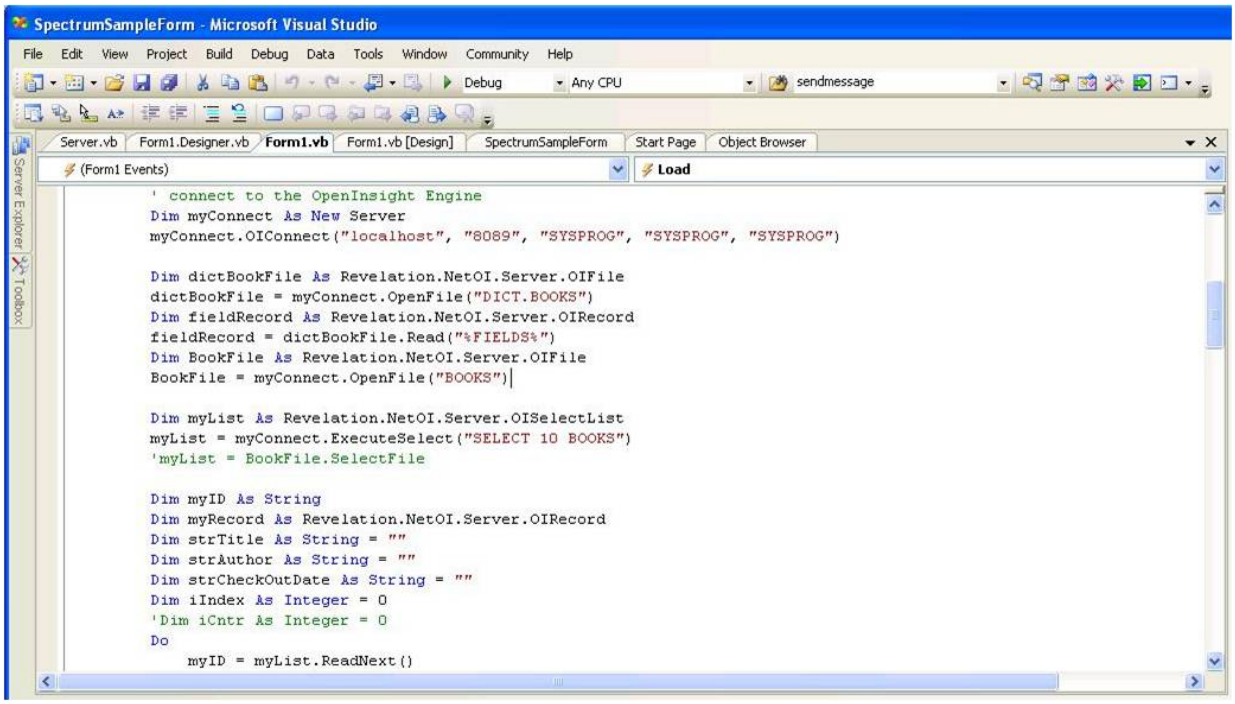

Figure 1: Using the NETOI classes in Visual Basic.NET

With this option, it is up to you, the developer, to decide where you want your logic to reside - either in Visual Studio (using OpenInsight to retrieve and update the records in your files), in OpenInsight (with your Windows application invoking the appropriate procedures in OpenInsight), or a combination of the two.

You can find sample NetOI code located in the NetOIExampleForm folder located in your OpenInsight directory.

Section II: The NetOI Assembly

The NetOI assembly contains the following classes:

Server

OIFile

OIRecord

OIKeyList

OIRecordList

The Server class contains the methods and properties used to connect to the OpenInsight database, and is the main class used for general NetOI calls.

NetOI can communicate with an OpenEngine (OEngine) either through the OpenInsight OEngineServer, or directly; the communication method chosen will depend on your particular needs. NetOI can also connect with a specific named OEngine, allowing NetOI and OpenInsight to "share" a single OEngine. Note however that in this situation, there is the potential for OpenInsight or NetOI to "block" processing; the NetOI server object contains additional methods to enable "out of band" (OOB) communications (via named pipes) to resolve this problem.

In order to establish a connection to the OpenInsight database, you must specify various application details (specifically, the application name, the user name, and the password for the application) as well as connection details (specifically, the location and port number for the EngineServer, or the location (as a windows path) of the oengine.dll). You can specify these individually, by setting the properties on your Server object, or you can make a single call to the OIConnect method and pass in these values at that time:

```
dim myConnect as Server = new Server
myConnect.OIConnect("localhost", "8088", "SYSPROG", "SYSPROG", "AAAAA")
```

Note that, if you specify a URL and port number, an EngineServer *must* be running for NetOI to communicate successfully with the OpenInsight database.

Once a connection has been established, you can invoke stored procedures or functions on the OpenInsight system directly, using the CallFunction or CallSub methods:

```
Dim theseParams(2) as string
theseParams(0) = "Value1"
theseParams(1) = "Value2"
theseParams(2) = "Value3"
dim theseResults() as string
theseResults = myConnect.CallFunction("MYFUNC", theseParams)
```

Note that the CallFunction and CallSub methods take their parameter lists as dimensioned string arrays (with a maximum of 10 parameters passed). Both methods return string arrays; the CallFunction returns the function return value, and all passed parameters, as its result, while the CallSub returns all the passed parameters as its result. In this way, if the OpenInsight procedure modifies any of the parameters, those changes are visible to the NetOI routine.

You can also "execute" a stored procedure through the Execute method. If the command you wish to execute will activate a select list, use the ExecuteSelect method instead. The ExecuteSelect method returns an OIKeyList object that you can process to retrieve all the selected keys.

The Server object is also responsible for handling Input and Output conversions, using the ICONV and OCONV methods:

```
Dim thisDate as string = myConnect.ICONV("04/22/2009", "D")
```

Finally, you can use various methods in the Server object to create OIFile, OIRecord, OIKeyList, and OIRecordList objects. To open a host file, you can call the OpenFile method, which returns an OIFile object:

```
dim myFile as OIFile = myConnect.OpenFile("BOOKS")
```

and, using an OIFile object, you can read and write records using the ReadRecord and WriteRecord methods:

```
dim myRecord as OIRecord = myConnect.ReadRecord(myFile, "1234")
```

and

```
myConnect.WriteRecord(myFile, "1234", myRecord)
```

You can select all, or some, of the keys in the file using the SelectFile method:

```
Dim theseKeys as NetOI.OIKeyList = thisOI.SelectFile(thisFile)
```

You can process through an OIKeyList in two different ways. Because an OIKeyList is "iterable", you can use the standard .NET "For Each" syntax:

```
for each thisKey as String in theseKeys
  ' process the key
Next
```

Alternatively, you can use OpenInsight Basic+-like syntax through the ReadNext method:

```
Dim thisKey as string
Do
  thisKey = theseKeys.ReadNext()
  if thisKey <> "" then
    ‘ process the key
  End if
Loop While theseKeys.ListStatus() = "1"
```

An OIKeyList is comparable to an activated "select list" in OpenInsight. NetOI also provides an OIRecordList, which in effect combines a ReadNext and a Read of the record. An OIKeyList can be turned into an OIRecordList with the RecordList method:

```
Dim thisFile as OIFile = thisOI.OpenFile("SAMPLE")
Dim theseKeys as OIKeyList = thisOI.SelectFile(thisFile)
dim theseRecords as OIRecordList = theseKeys.RecordList(thisFile)
```

Once an OIRecordList has been created, it can be iterated through using either of the same methods described above, for example:

```
for each thisRecord as OIRecord in theseRecords
  ' process the record
Next
```

Note, however, that an OIRecordList consists of OIRecord objects, whereas an OIKeyList consists of strings (which are just the "key fields" of the records).

An OIRecord allows you to manipulate and extract OpenInsight record contents in a variety of ways. The entire contents can be retrieved (as "raw" data, in internal format) using the Record property:

```
dim strAllFields() as string = thisRecord.Record
```

Each field of the record is returned as an element in a string array. You can similarly set the entire contents of an OIRecord by passing in a string array to the Record property.

To return or set the data as an XML string (again, in "raw" format), you can use the XMLRaw property.

If your OIRecord has been read from an OIFile, and that OIFile was opened with both a dictionary and data section (the default behavior), then you can extract or set information in the OIRecord using the dictionary names:

```
Dim strFirstName as String = thisRecord("FIRST_NAME")
thisRecord("ENTRY_DATE") = "04/22/2009"
```

Any conversions are applied appropriately. Note that only F-type dictionary items are available for your use in this fashion.

You may also specify a field number:

```
Dim strFirstName as string = thisRecord(2)
```

Note that when a field number (as opposed to a field name) is used, conversions are not applied automatically. To extract or set individual values or subvalues, you may use the Field property; for example, to extract the 2nd value of the 3rd field:

```
Dim value2 as String = thisRecord.Field(3,2)
```

An advanced feature of NetOI is the ability to set or get all, or a subset, of the OIRecord contents as an XML string. While the XMLRaw property gets or sets the entire contents of the OIRecord in "raw" format, the XML property automatically performs any input or output conversions as required. To use the XML property, you must first establish which fields you wish the XML to contain, using the AddDefaultDict property on the OIFile that the records will belong to:

```
Dim thisFile as Server.OIFile
thisFile = thisOI.OpenFile("SAMPLE")
thisFile.AddDefaultDict("FIRST_NAME")
thisFile.AddDefaultDict("LAST_NAME")
thisFile.AddDefaultDict("CITY")
```

Once the list of default dictionaries has been specified, you can retrieve the contents of any OIRecord in this XML format:

```
Dim thisRecord as OIRecord = thisFile.Read("ABC")
Dim xmlString as String = thisRecord.XML()
```

As a full example of how NetOI works, we can create a sample VB.NET form to display some file content in a ListView control.

First, create a new VB.NET project, and add a reference to NetOI. Next, add the NetOI assembly into your imports, and define a new ListView and new Server objects:

```
Imports Revelation.NetOI
Public Class Form1
   Dim listView1 As New ListView
   Dim myConnect As New Server
```

On the form load event, add the code to initially create the listView, and connect to the OpenInsight database:

```
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load

   listView1.Bounds = New Rectangle(New Point(10, 10), New Size(500, 300))
   ' Set the view to show details.
   listView1.View = View.Details
   ' Allow the user to edit item text.
   listView1.LabelEdit = True
   ' Allow the user to rearrange columns.
   listView1.AllowColumnReorder = True
   ' Display check boxes.
   listView1.CheckBoxes = False
   ' Select the item and subitems when selection is made.
   listView1.FullRowSelect = True
   ' Display grid lines.
   listView1.GridLines = True
   ' Sort the items in the list in ascending order.
   listView1.Sorting = SortOrder.Ascending

   ' connect to the OpenInsight Engine
   myConnect.PersistentConnection = True
   myConnect.OIConnect("localhost", "8088", "SYSPROG", "SYSPROG", "xxxxx")
```

Open up the “BOOKS” table in the OpenInsight database, and select some keys. Note, in the commented out lines, different ways to select the records (including manually creating an OIKeyList from a string array):

```
   ' Open the dict & data section of the BOOKS file
   Dim BookFile As Revelation.NetOI.OIFile
   BookFile = myConnect.OpenFile("BOOKS")

   ' get a list from the BOOKS file
   Dim myList As Revelation.NetOI.OIKeyList
   ' a list based on a select statement
   'myList = myConnect.ExecuteSelect("SELECT 10 BOOKS")
   ' a list selecting the whole file
   'myList = BookFile.SelectFile
   ' a list selecting 20 records from the file
   myList = BookFile.SelectFile("20")

   ' or manually create a list of keys
   'Dim myKeys(1) As String
   'myKeys(0) = "105"
   'myKeys(1) = "ABCD"
   'myList = myConnect.MakeList(myKeys)
```

Process through the selected items (after converting the OIKeyList into an OIRecordList), and populate the ListView object:

```
   Dim strTitle As String = ""
```

```
Dim strAuthor As String = ""
Dim strCheckOutDate As String = ""
Dim myID As String = ""

' turn the select list into a list of records
Dim myRecords As OIRecordList = BookFile.RecordList(myList)

' walk through the record list
For Each myRecord As OIRecord In myRecords
   ' pull out the key field
   myID = myRecord.Key()
   strTitle = myRecord("TITLE")
   strAuthor = myRecord("AUTHOR")
   strCheckOutDate = myRecord("CHECK_OUT_DATE")
   ' add this information into the ListView control
   Dim thisItem As New ListViewItem(myID, iIndex)
   ' alternate images
   If iIndex = 0 Then
      iIndex = 1
   Else
      iIndex = 0
   End If
   thisItem.SubItems.Add(strTitle)
   thisItem.SubItems.Add(strAuthor)
   thisItem.SubItems.Add(strCheckOutDate)
   listView1.Items.Add(thisItem)
Next
```

The rest of the code finishes up the ListView creation:

```
' Create columns for the items and subitems.
listView1.Columns.Add("Book ID", -2, HorizontalAlignment.Left)
listView1.Columns.Add("Title", -2, HorizontalAlignment.Left)
listView1.Columns.Add("Author", -2, HorizontalAlignment.Left)
listView1.Columns.Add("Check-out Date", -2, HorizontalAlignment.Center)

' Create two ImageList objects.
Dim imageListSmall As New ImageList()
Dim imageListLarge As New ImageList()

' Initialize the ImageList objects with bitmaps.
imageListSmall.Images.Add(Bitmap.FromFile("C:\revsoft\oi90gold\bmps\arev32(SMALL).bmp"))
imageListSmall.Images.Add(Bitmap.FromFile("C:\revsoft\oi90gold\bmps\OI32.bmp"))
imageListLarge.Images.Add(Bitmap.FromFile("C:\revsoft\oi90gold\bmps\arev32(BIG).bmp"))
imageListLarge.Images.Add(Bitmap.FromFile("C:\revsoft\oi90gold\bmps\RTI.bmp"))
'Assign the ImageList objects to the ListView.
listView1.LargeImageList = imageListLarge
listView1.SmallImageList = imageListSmall

' Add the ListView to the control collection.
Me.Controls.Add(listView1)
```

Although not required, you may close any open files using the Close method, and you may disconnect your OpenInsight connection using OIDisconnect:

```
BookFile.Close()
```

```
        ' close our connection to OI
        myConnect.OIDisconnect()

    End Sub

End Class
```

Section III: NetOI Configuration

NetOI has the ability to process many common output conversion codes locally, rather than on the host. The supported OCONV codes include numeric (MD, MC), date (D), time (MT), datetime (DT), character (MCU, MCT, MCL, MCX, MCO, MCB, HEX) and binary (B). This behavior is currently disabled by default, however.

The user may choose to activate this behavior by editing the record CFG_NETOI in the SYSENV table. If field 1 of this record is "0" (the default), then NetOI will *not* process the conversion codes locally, but will continue to rely on the host for all OCONV processing. If field 1 of this record is set to "1", then NetOI *will* process the available conversion codes locally (and not on the host).

In addition, in order to ensure that the conversion processing is done correctly, NetOI performs a "startup check" when it is first called upon to do an output conversion. It validates that the results returned from the host for a sample date, time, numeric, and Boolean match those generated locally; if they do not, then NetOI continues to rely on the host for processing, even if CFG_NETOI field 1 is set to "1". (Note: in certain circumstances, it may be desirable to override the startup check and *force* NetOI to use its own local processing for the available OCONV codes. Set field 1 of the CFG_NETOI record to "-1" to force local processing of the conversion codes regardless of the results of the startup check).

Also note that this behavior can be limited to a particular user in an application by creating or updating the record CFG_NETOI*<appid>*<username> in SYSENV, or to a particular user in any application (by creating or updating the record CFG_NETOI**<username>), or to all users in a particular application (by creating or updating the record CFG_NETOI*<appid>).

REVELATION
SOFTWARE

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.
45 St Mary's Road
Ealing,
London, W5 5RG
U.K.
Phone: +44 0 208 912 1000
Fax: +44 0 208 912 1001
info@revsoft.co.uk

Revelation Software Australia Pty Ltd.
PO Box 300
Brookvale, NSW 2100
Australia
Phone: +61 2 8003 4199
Fax: +61 2 9332 6099
info@revelationsoftware.com.au

Revelation Software is a division of Revelation Technologies, Inc.

Part No. 59-931