Implementing Notification Handlers with the DM Extensions API.

The purpose of this document is to demonstrate the use of Notification Handlers to extend the functionality of eDOCS DM Extensions. Notifications are Event handlers that are provided through the Event and Notification handlers. There is a subtle difference between an Event handler and a Notification handler. A Notification handler sends a notification for pre-defined events that can be extended. An Event handler is limited to providing functionality that is well defined.

Notifications are useful for extending functionality to events that occur. For example, instead of writing a new menu command with the DM API constructs to put a string into a document name as an Event handler, a Notification handler can be written to put a string in the document name prior to a Save method being invoked. Notification handlers are triggered by methods that occur to only Profile objects and Version objects. Notification handlers can also be defined prior to invoking a method and after a method has finished processing. For example, Notification handlers can be written to display a message to the user prior to and after uploading a file into DM. A notification is sent before and after the UploadFile method is invoked. As a result, many useful extensions can be written to extend existing functionality.

The initial steps for creating a Notification handler are similar to creating an Event handler. In fact, they virtually the same. The IOMEventHandler interface must be supported. However, instead of implementing the IOMDocumentMenuSink interface, the IOMDocumentEventSink interface must be implemented. The IOMDocumentEventSink interface is implemented so that it is possible to receive notifications regarding Profile object events and Version object events

The IOMDocumentEventSink\_ProfileEvent method is used to handle all notifications for methods that occur to Profile objects. The IOMDocumentEventSink\_VersionEvent method is used handle all notifications for methods that occur to Version objects

**IOMDocumentEventSink\_ProfileEvent**

The following method is used to handle notifications that occur for Profile objects:

short ProfileEvent(short sEventID, short sEventType, IProfile pProfile, IBCCoreCtx pParams)

* Must be implemented as a Function and return a value from the HResultCodes enumeration

HS\_NEXT\_HANDLER – Default to move to next handler

HS\_CANCEL\_NOTIFY – Cancels chain of events

HS\_EVENT\_PROCESSED – Stops processing events

− pEventID represents the event which is a value from the [DEvents](#DEvents) enumeration

− pEventType represents the type of event which is a value of the [DEventType](#DEventsTypes) enumeration

ET\_PRE before the method gets executed

ET\_POST after the method gets executed

− pProfile represents the handle to the Profile object. Though passed by value, methods can still be invoked on the Profile object

− pParams represents any extra data that is sent as part of the parameters to the method that was invoked

DE\_LOCK event has the ReturnDate, Flags, and Comments that can be used as keys to retrieve extra information from the BCCoreCtx object

**IOMDocumentEventSink\_VersionEvent**

short VersionEvent(short sEventID, short sEventType, IVersion pVersion, IBCCoreCtx pParams)

* Must be implemented as a Function and return a value from the HResultCodes enumeration

HS\_NEXT\_HANDLER – Default to move to next handler

HS\_CANCEL\_NOTIFY – Cancels chain of events

HS\_EVENT\_PROCESSED – Stops processing events

− pEventID represents the event which is a value from the [DEvents](#DEvents) enumeration

− pEventType represents the type of event which is a value of the [DEventType](#DEventsTypes) enumeration

ET\_PRE before the method gets executed

ET\_POST after the method gets executed

− pVersion represents the handle to the Version object. Although passed by value, methods can still be invoked on the Version object

− pCtx represents any extra data that is sent as part of the parameters to the method that is invoked DE\_CHECK\_OUT event has the ReturnDate, Flags, and Comments that can be retrieved

**Creating a sample Notification Handler**

The steps to create a sample notification handler are as follows:

**Step 1 – Create a Project in Microsoft Visual Studio**

1. Start Microsoft Visual Studio.

2. On the File menu, click New Project. The New Project dialog box appears.

3. In the Project Types section, expand Visual C#, and then click Windows.

4. In the Templates section, click Class Library.

5. In the Name field, enter a name for your project. (For this example, the project is named NotificationHandler.)

6. Click OK.

**Step 2 – Add Component References to the Project**

To add the necessary component references to your project, complete the following steps:

1. In the Solution Explorer pane, right-click the References node, and then click Add Reference.

2. In the Add Reference dialog box, click the Browse tab, and then browse to the Open Text\DM Extensions\ folder.

3. Select the following component files:

Hummingbird.DM.Extensions.Interop.DECore

Hummingbird.DM.Extensions.Interop.DOCSObjects

Hummingbird.DM.Extensions.Interop.VHELPER

4. Click OK.

5. Right-click the References node in Solution Explorer, and then click Add Reference.

6. In the Add Reference dialog box, click the Assemblies >>Framework tab, and then select the System.Windows.Forms component.

7. Click OK.

8. At the top of the Class tab, add the following using statement:

|  |
| --- |
| using System.Windows.Forms; |

**Step 3 - Implement Interfaces**

1. In the Class tab, right-click Class1, click Refactor, and then click Rename.

2. Enter a unique name for your class, and then click OK. (For this example, the name used is Example.)

3. In the Preview Changes – Rename dialog box, review the changes, and then click Apply.

4. Implement the IOMEventHandler and IOMDocumentMenuSink interfaces by changing the class signature to the following:

|  |
| --- |
| public class Example : Hummingbird.DM.Extensions.Interop.DOCSObjects.IOMEventHandler, Hummingbird.DM.Extensions.Interop.DOCSObjects.IOMDocumentEventSink |

5. Right-click the IOMEventHandler interface, and then click Implement Interface > Implement Interface.

6. Right-click the IOMDocumentMenuSink interface, and then click Implement Interface > Implement Interface.

You should now see the following events added to the class:

public void Init(IApplication pApp)

{

throw new NotImplementedException();

}

public void Terminate(IApplication pApp)

{

throw new NotImplementedException();

}

public short ProfileEvent(short sEventID, short sEventType, IProfile pProfile, IBCCoreCtx pParams)

{

throw new NotImplementedException();

}

public short VersionEvent(short sEventID, short sEventType, IVersion pVersion, IBCCoreCtx pParams)

{

throw new NotImplementedException();

}

}

**Step 4 - Add Code**

1. Add the following as class level variables:

private Hummingbird.DM.Extensions.Interop.DOCSObjects.Application dmApp;

private Hummingbird.DM.Extensions.Interop.VHELPER.VBConnector connector;

1. Explicitly define a constructor for your class as follows:

public Example()

{

connector = new Hummingbird.DM.Extensions.Interop.VHELPER.VBConnector();

dmApp = new Hummingbird.DM.Extensions.Interop.DOCSObjects.Application();

}

3. In the Init function, add the following connector:

int interfacesConnected = connector.Connect(pApp, this);

4.In the Terminate event, remove throw new NotImplementedException();

5..Add the following code to the Terminate event for proper cleanup:

|  |
| --- |
| System.Runtime.InteropServices.Marshal.FinalReleaseComObject(dmApp);  System.Runtime.InteropServices.Marshal.FinalReleaseComObject(connector);  dmApp = null; |

5. In the ProfileEvent function add the following code:

if (sEventID == (short)Hummingbird.DM.Extensions.Interop.DOCSObjects.DEvents.DE\_OPEN)

MessageBox.Show("Document " + pProfile.DocNumber.ToString() + " opened");

return (short)HResultCodes.HS\_NEXT\_HANDLER;

**Step 5 – Configure the Build Options**

1. In the Solution Explorer pane, right-click the project name, and then click Properties.

2. In the Solution Property Pages dialog box, click the Application tab, and then click the Assembly Information button.

3. Select the Make assembly COM-Visible check box, and then click OK.

**Step 6 - Save and Build**

1. Save your solution.
2. Right-click the solution in the **Solution Explorer** pane, and then click **Build Solution**.

**Step 7 – Deploy**

1. Copy your resulting .dll file into the DM Extensions folder on your client.
2. Use Regasm to register your .dll from a cmd prompt

Example: C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe "C:\Program Files (x86)\Open Text\DM Extensions\NotificationHandler.dll"

1. Create the following registry key

[HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\Hummingbird\PowerDOCS\OM\EventHandlers\NotificationHandler]

@=" NotificationHandler.Example"

The format for the required registry key is @="<Namespace>.<ClassName>"

**DEvents Enumeration Description**

DE\_OPEN: Indicates that a document is being opened in its native application with the Open method or version is being opened in its native application with the Open method.

DE\_OPEN\_COMPATIBLE: Indicates that a document is being opened in an application compatible with its native application with the OpenInCompatibleApp method or a version is being opened in an application compatible with its native application with the OpenInCompatibleApp method.

DE\_LOCK: Indicates that a document profile is being locked with the LockProfile method.

DE\_UNLOCK: Indicates that a document profile is being unlocked with the UnLockProfile method.

DE\_DELETE: Indicates that a document is being deleted with the Delete method.

DE\_PRINT: Indicates that a document is being printed with the PrintDoc method.

DE\_PREVIEW: Indicates that the documents are being viewed with the Preview method.

DE\_SELECT\_FOR\_EDIT: Indicates that a document is being selected for editing with the SelectForEdit method.

DE\_RELEASE\_DOC: Indicates that a document is being unlocked with the ReleaseDoc method.

DE\_SAVE: Indicates that a document profile is being savedwith the Save method.

DE\_NEW\_DOC: Indicates that a new profile is being created with the CreateProfile method on a library.

DE\_NEW\_VERSION: Indicates that a new version has been createdusing the CreateVersionFromFile method.

DE\_ADD\_LINK: Indicates that a new item has been added to the Folder using the AddDocument or AddSubProject methods.

DE\_REMOVE\_LINK: Indicates that an item has been removed from a Folder using the RemoveDocument or RemoveSubProject methods.

DE\_UPDATE\_LINK: Indicates that an item in a Folder has been updated using ChangeDisplayNameForm or ChangeVersionTypeForm.

**DEventTypes Enumeration Description**

ET\_PRE: Indicates that the method is called before the event occurs and the corresponding Profile or Version method is processed.

ET\_POST: Indicates that the method is called when the event has occurred and the corresponding Profile or Version method has been processed