**Microsoft Office SharePoint Server (MOSS) for Developers**

**(Part 1)**

**Topic 3: Programming Event Handlers**



[**http://msdn.microsoft.com/rampup**](http://msdn.microsoft.com/rampup)

**Introduction**

Windows SharePoint Services 2.0 introduced sinking events that were triggered from within a Windows SharePoint Services 2.0 document library by attaching a Microsoft .NET assembly to the document library. The event sinking was limited, however, to the document library or form library, and events were triggered only after the actual action occurred within the library. Windows SharePoint Services 3.0 goes a step further and introduces a new and extensive set of events that you can use, not only within libraries but also now within lists, sites, and user operations.

Events in Windows SharePoint Services 3.0 now support the following capabilities:

* You can use pre-synchronous and post-asynchronous modes to handle events.
* You can register events at the file, list, and Web site levels.
* You can register multiple events per item.
* You can cancel pre-synchronous events and display custom error messages.
* You can register item events on generic lists, as well as on document libraries.
* You can declaratively register events per list type or content type.
* In addition to trapping standard list item events (for example, [ItemAdding](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemadding.aspx), [ItemDeleting](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemdeleting.aspx), or [ItemUpdating](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemupdating.aspx)), you can trap schema events (such as adding, removing, or updating columns), as well as Web site and site collection deletion.

**Creating a Basic Event Handler**

The following example shows the basic steps of creating an event handler that executes code before a list item is deleted or after an item is added. The example works on announcement lists, adding text to the body of new items, but cancelling attempts to delete existing items.

You create an event handler assembly in Microsoft Visual Studio 2005 by creating a class library. You add the reference to Microsoft.SharePoint.dll and inherit from the [Microsoft.SharePoint.SPItemEventReceiver](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.aspx) base class.

Windows SharePoint Services 3.0 instantiates an object of your class when actions occur within the list. As a developer, you override the methods of the SharePoint base class for the events you are interested in handling. The methods of the **SPItemEventReceiver** class that you can override are the following:

* ContextEvent
* ItemAdded
* ItemAdding
* ItemAttachmentAdded
* ItemAttachmentAdding
* ItemAttachmentDeleted
* ItemAttachmentDeleting
* ItemCheckedIn
* ItemCheckedOut
* ItemCheckingIn
* ItemCheckingOut
* ItemDeleted
* ItemDeleting
* ItemFileConverted
* ItemFileMoved
* ItemFileMoving
* ItemUncheckedOut
* ItemUncheckingOut
* ItemUpdated
* ItemUpdating

The example overrides two methods: **ItemDeleting** and **ItemAdded**. Remember that the suffix "ing" indicates that we are handling the event in a synchronous way before the action actually occurs, and the suffix "ed" indicates that we are handling the event in an asynchronous way after the action occurs.

You have access to the data associated with the [SPListItem](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.splistitem.aspx) object through an object that is provided as an input parameter when your method is called. This object is of type [SPItemEventProperties](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventproperties.aspx). Through the Cancel property of the **SPItemEventProperties** class you can cancel the event, and through the ErrorMessage property you can display a custom error message on a SharePoint page.

The second method to override is **ItemAdded**. The following example uses the [ListItem](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventproperties.listitem.aspx) property to return an object that represents the new list item, and then modifies the body text of the item.

You must register both the **ItemDeleting** and **ItemAdded** event receivers as described in [Registering an Event Handler](http://msdn.microsoft.com/en-us/library/ms475328.aspx).

**Registering an Event Handler**

Event handlers are registered differently in Windows SharePoint Services 3.0 because of new concepts like content types and Features that it introduces. For backward compatibility reasons, however, Windows SharePoint Services 3.0 supports the existing registration of library events. The **EventSinkAssembly**, **EventSinkClass**, and the **EventSinkData** properties continue to function and are exposed in the user interface.

In Windows SharePoint Services 3.0, there are three fundamental ways to register an event handler:

* Through the object model, as the [SPWeb](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spweb.aspx) and [SPList](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.splist.aspx) classes now each provide an **EventReceivers** property through which to access the collection of event receiver definitions for the Web site or list. You can add new event receivers by calling the [Add](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.speventreceiverdefinitioncollection.add.aspx) method.
* Declaratively by list type, for example, to register an event handler for all announcements lists. In a Feature.xml file, you can register an event handler by list template ID. When the containing feature is activated per **SPWeb** object, you can register the event handler for any list of the specified type.
* Declaratively by content type, for example, to register an event handler for all documents of a specific type. Within the XML for a content type definition, you can register event receivers.

**Note** The assembly containing your event handler must be strongly named and registered in the global assembly cache (GAC) to be used. You cannot operate event receiver assemblies from the \\_app\_bin folder (*Local\_Drive*:\Inetpub\wwwroot\wss\VirtualDirectories\*GUID*\\_app\_bin).

**Content Types**

Windows SharePoint Services 3.0 introduces the concept of content types in the data store. In short, content types introduce the notion of reusability to Web designers working with Windows SharePoint Services. Web designers now have the ability to create classes of objects with specific definitions and possible associated behavior, such as type name, fields, format, business processes, retention, auditing, and event handling. You can now activate SharePoint lists and libraries to support multiple content types. When you do that, you can attach one or more of these classes to your list or library and thus extend it with additional functionality and behavior. Think of extending a customer list with a Contact content type. The Contact content type can provide the Customer list with a set of new fields, like Contact Name, Function, Telephone, and so forth, and also with new behavior.

You can now define event handlers for a specific content type. You can, for example, define the content type "Customer", and within its behavior, you can define the metadata for the event handler.

**Features**

You define content types by using a feature. When you define a content type with a feature, you create two XML files as shown below:

* **Feature.xml**   You use this XML file to define the metadata for the new feature. The following example code scopes the feature at the level of the site and defines a unique identifier for the new feature. Using the **ElementManifests** element, it then points to the location of the second XML file storing all of the detailed information on the feature itself.
* **Elements.xml**   You use this file to define the assembly that encapsulates the event handler, the class itself, and also a sequence number that specifies the order, if multiple event handlers are associated with the feature. The following example registers event receivers for deleting and adding items.

**Event Troubleshooting**

Programming events can cause exceptional behaviors, depending on the contexts in which you implement the event handlers. The following table describes event behaviors related to specific contexts that you might encounter when writing code.

|  |  |
| --- | --- |
| **Context** | **Description** |
| Document libraries and content types | You create an .aspx page of a particular content type in a document library, and the page has associated [ItemAdding](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemadding.aspx) and [ItemAdded](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemadded.aspx) events, but the events do not fire.  Only default content type properties are present in the Add phase. Additional properties (fields) of the specified content type are added only in the Update phase. Therefore, best practice is to avoid firing **Add** events with a specified content type ID. Instead, set the content type to the default content type in the Add phase. The content type can subsequently be changed in the Update phase. An alternative is to register the events at the list level. |
| Document libraries and content types | An **ItemAdding** event that is bound to a content type fires even when a document that is not of that type is uploaded.  When you first add a document to a library, the content type is always the default content type associated with the list. The content type of the document can be changed during the Update phase and the content type ID is adjusted accordingly. |
| Document libraries and content types | A request to delete an item through a list form does not have an associated content type ID, but this causes an [ItemDeleting](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemdeleting.aspx) or [ItemDeleted](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemdeleted.aspx) event to fire on all items, not just on items of a specific content type, even though the event was registered only for the content type.  By design, Windows SharePoint Services fires events for all items in a list when the request is not bound to a content type to allow firing an event for all items when the event receiver is registered for all items in the list.  This behavior affects policies that involve **Delete** events. If you implement a policy that involves deletion, apply the policy to a content type, and then bind the content type to a list, the policy will apply to all items in the list, not only to the items of the content type to which the policy is applied. |
| Lists and content types | You register [ItemUpdating](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemupdating.aspx) and [ItemUpdated](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventreceiver.itemupdated.aspx) events on a content type that is bound to a list, but the events fire even when items that are not of that content type are updated through the object model.  Windows SharePoint Services returns 0 (zero) as the content type ID, not the content type ID of the item. |
| Lists | The [BeforeProperties](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventproperties.beforeproperties.aspx) property applies only to **DocumentLibrary** type lists.  The workaround is to return the properties by using the object model and the given list item ID. |
| Document libraries | You add a custom content type to a document library. **ItemAdding**, **ItemAdded**, and **ItemUpdating** events fire, but not **ItemUpdated** events.  This behavior occurs only for the Shared Documents document library and not for custom document libraries. For Shared Documents, adding a content type creates a Forms/<Content Type> folder and copies a Template.doc file into that folder, which causes the events to fire. |
| Document libraries and content types | For folders, you get the content type and content type ID through the [AfterProperties](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spitemeventproperties.afterproperties.aspx) property in the **ItemAdding** and **ItemAdded**, events, but for documents, you get the content type ID only in the **ItemAdded** event and nothing in the **ItemAdding** event.  The Windows SharePoint Services parser for Microsoft Office system documents does not set the content type name in a document dictionary. Document metadata does not set a content type name. |
| Lists | List events do not fire on the UserInfo list. |
| Lists | An **ItemAdded** event fires on a folder type item, but the list item ID is not returned. |
| Document libraries | Events do not fire when unlinking a document copy from the original document. |
| Document libraries | List item IDs always equal 0 when events fire on documents in the Forms folder. If you register an **ItemUpdating** event in a document library and then modify EditForm.aspx, the event fires, but the ID equals 0 (zero) because the item is not a regular list item. |
| Lists | Event receivers do not bind to a list when the list is provisioned through [Onet.xml](http://msdn.microsoft.com/en-us/library/ms474369.aspx) and the list type has an associated Feature that binds a receiver to the list. You create a Feature that binds a receiver to a list type, create a list of that type through the site definition, but the receiver is not bound to the list when a site is provisioned. A workaround is to bind the receiver to a content type instead, and then bind the content type to the list. |