Colligo Contributor SDK

Custom Editor

CONTRIBUTOR CUSTOM EDITOR

Colligo Contributor applications support creation of Custom Editors. Custom Metadata Editors allow businesses to enhance user experience by:

- Integrating with corporate look and feel
- Rearranging layouts to provide intuitive user-interface
- Accessing data from external sources; for example: Metadata-values may be drawn from an external LOB data-source (CRM) or web-service
- Applying business rules to data input; for example: Cascading lookups, a user would select **State** from a lookup field and the **City** field is filtered

Implementing and Installing Custom Editors

Custom Editors are contained in a single.Net assembly. The .Net assembly should contain a public Type that implements the Colligo.WML.MetaData.ICustomEditor interface. ICustomEditor is defined in the Colligo.WML assembly which is installed in the Global Assembly Cache when Contributor is installed.

To register the Custom Editor assembly with Contributor define two new string values within the registry:

Key Name: HKEY_LOCAL_MACHINE\SOFTWARE\ColligoOfflineClient\UserInterface\MetaDataEditor
Value1: AssemblyPath (REG_SZ)
Value2: EditorClass (REG_SZ)

AssemblyPath: The full path to the assembly containing the Custom Editor class
Eg: C:\Program Files\Colligo Networks\Colligo Contributor 3.0\CustomEditor.dll

EditorClass: The FullName of the Type that implements the ICustomEditor interface
Eg: Contoso.Contributor.CustomEditor

On startup, Contributor reads the registry and loads the Custom Editor assembly. If the assembly is not found, or the specified Type does not implement the ICustomEditor interface, Contributor logs an exception and continues using the Default Editor.

Custom Metadata Editor Modes

A Custom editor displays in three different modes:

- Create
- View
- Edit

Each mode may be invoked for a single item or multiple items. The Custom Editor should be designed to handle these situations.

Resolve Conflict Editor

Custom Editors are also used to resolve the Conflict sync issue. The Resolve Conflict Custom Editor is implemented by the same class as the Custom Metadata Editor. The Resolve Conflict editor can present the conflicting items side-by-side and aid the conflict resolution process according to your business rules.

The ICustomEditor interface is shown below:

```
namespace Colligo.WML.MetaData
{
    public interface ICustomEditor
    {
        EditorResult ShowEditor(IEditContext context);
        ResolveConflictResult ShowResolveConflictDialog(IResolveConflictContext
context);
    }
}
```

ICustomEditor. ShowEditor

When Contributor needs to display Metadata editor, it checks whether a Custom Editor has been loaded. If it has, it calls the ShowEditor function, passing an IEditContext object. The IEditContext provides the editor with access to:

- The item (or items) being edited and, for libraries, their associated files
- List or Library and folder containing the item(s)
- The Edit Mode: Create; Edit; or View

ShowEditor() return one of the EditorResult enumerated types. Possible values for EditorResult are:

- **OK**: The Custom Metadata editor exited with ZOK
- **Cancel**: The Custom Metadata editor exited with **Cancel**
- **UseDefaultEditor**: The Custom Editor is not used for this context. Use the Default Editor.

```
namespace Colligo.WML.MetaData
{
    public interface IEditContext
    {
        IContentType ContentType { get; }
        IDocInfo[] DocInfos { get; }
        EditorMode EditorMode { get; }
        IList List { get; }
        IListItem[] ListItems { get; }
        IWin32Window Parent { get; }
        IListItem ParentFolder { get; }

        void BroadcastItemUpdate(IListItem listitem);
        void BroadcastListUpdate(IList list);
        ItemCommitResult CommitItem(IListItem listitem, IListItem parentfolder,
IList list, IDocInfo docinfo);
        bool RetrieveDefaultProperties(IListItem listitem, IList list, IDocInfo
docinfo, IContentType contenttype);
    }
}
```

IEditContext Properties

Property	Description	Condition
ContentType	The user clicked **New > ContentType** in Contributor	Only supplied on a Create operation
DocInfos	The DocInfo provides information about the file that are being worked on	Applicable to Document Libraries
EditorMode	The Editor is being displayed for Create; Edit; or View	
List	The List containing the item(s)	
ListItems	Items themselves; the ListItems contain the metadata values	
Parent	Parent Window	
ParentFolder	The folder containing the items	Only supplied on a Create operation

IEditContext Functions

Function	Behavior
RetrieveDefaultProperties	Retrieves the SharePoint-specified default properties for the specified item
CommitItem	Commits changes to a specific item
BroadcastItemUpdate(IListItem listitem)	Notifies the Contributor UI that a specific item has been updated
BroadcastListUpdate(IList list)	Notifies the Contributor UI that items in the specified list have been updated

Typical Custom Metadata Editor Procedure

The Custom Metadata procedure typically follows a 5-step process:

1. Pre-Initialize Check: The current context is a supported context. This may involve checking the Target List; EditMode, Number of Items; Content-Type; etc. Return UseDefaultEditor for unhandled context(s).
2. Initialize: For the item(s) call RetriveDefaultProperties to get the SharePoint specified detail values.
3. Handle-Edit: For each of the items process the metadata using SetProperty calls.
4. **Complete Use CommitItem** for each item in the context to commit the changes to Contributor Notify the Contributor UI that item(s) have changed. For a single-item edit use BroadcastItemUpdate(IListItem listitem) For a multi-item edit use BroadcastItemUpdate(IList list)
5. **Return Complete the operation - returning EditResult.OK to Contributor**. Note: No further processing is applied when EditResult.OK, or EditResult.Cancel are returned. If EditResult.UseDefaultEditor is returned Contributor will display its Default Editor.