What do you want your website to do?

ektron

Starter Site API Reference Documentation

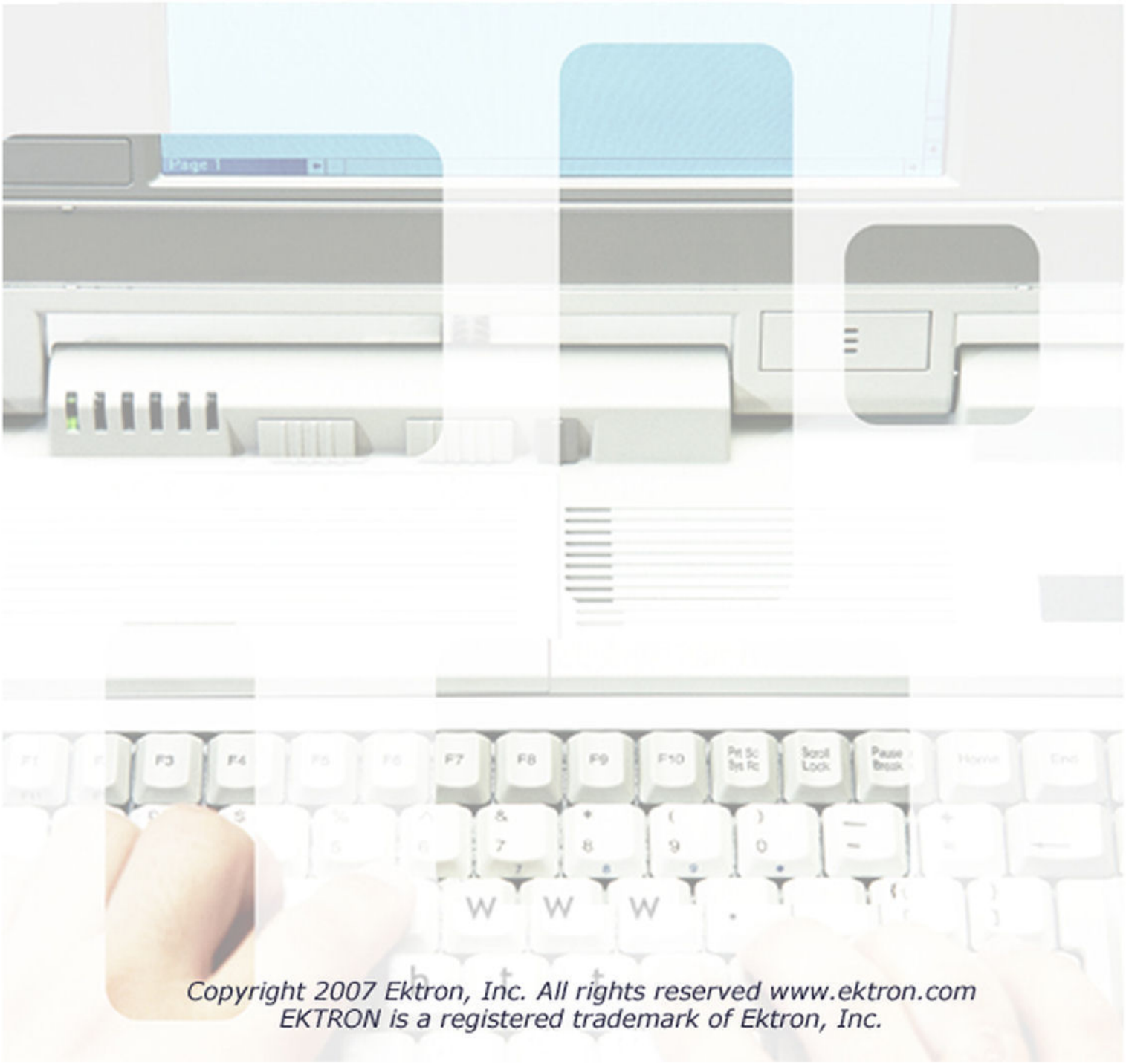


Table of Contents

Ektron Namespace

This is namespace Ektron.

Namespaces

Name	Description
Cms (see page 1)	This is namespace Ektron.Cms.

Ektron.Cms Namespace

This is namespace Ektron.Cms.

Namespaces

Name	Description
StarterApps (see page 1)	This is namespace Ektron.Cms.StarterApps.

Ektron.Cms.StarterApps Namespace

This is namespace Ektron.Cms.StarterApps.

Namespaces

Name	Description
CaseManagement (see page 1)	This is namespace Ektron.Cms.StarterApps.CaseManagement.
ProjectManagement (see page 66)	This is namespace Ektron.Cms.StarterApps.ProjectManagement.
Wiki (see page 130)	This is namespace Ektron.Cms.StarterApps.Wiki.

Ektron.Cms.StarterApps.CaseManagement Namespace

This is namespace Ektron.Cms.StarterApps.CaseManagement.

Classes

	Name	Description
	CaseManagement (see page 2)	Contains methods and properties for the overall creation and management of the Case Management application. Important methods worth highlighting are **Initialize (see page 16)** and **GetInformationArchitecture (see page 10).** Initialize (see page 16) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 10) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	Cases (see page 26)	Contains methods and properties that manage the Case aspect of the Case management site. This includes functionality for Blogs, Discussion Boards, Taxonomy, Wiki (see page 130) and Folders.
	Clients (see page 46)	Contains methods for managing clients in the Case Manager Starter Application.
	Users (see page 51)	Contains methods for managing users in the Case Manager Starter Application.
	XMLUtilities (see page 58)	Contains methods and properties for handling the XML used within the Case Manager Starter Application.

Legend

	Class

Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	CaseManagement (see page 2)	Contains methods and properties for the overall creation and management of the Case Management application. Important methods worth highlighting are **Initialize (see page 16)** and **GetInformationArchitecture (see page 10).** Initialize (see page 16) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 10) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	Cases (see page 26)	Contains methods and properties that manage the Case aspect of the Case management site. This includes functionality for Blogs, Discussion Boards, Taxonomy, Wiki (see page 130) and Folders.
	Clients (see page 46)	Contains methods for managing clients in the Case Manager Starter Application.
	Users (see page 51)	Contains methods for managing users in the Case Manager Starter Application.
	XMLUtilities (see page 58)	Contains methods and properties for handling the XML used within the Case Manager Starter Application.

Legend

	Class

CaseManagement Class

Contains methods and properties for the overall creation and management of the Case Management application. Important methods worth highlighting are **Initialize (see page 16)** and **GetInformationArchitecture (see page 10).** Initialize (see page 16) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 10) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.

Class Hierarchy

Ektron.Cms.StarterApps.CaseManagement.CaseManagement

C#

```
public class CaseManagement;
```

Visual Basic

```
Public Class CaseManagement
```

File

CaseManagement.vb

CaseManagement Enumerations

The enumerations of the CaseManagement class are listed here.

Enumerations

	Name	Description
	InformationArchitectureUserModes (see page 3)	**MembershipUser** - can add members and Case to clients. They cannot remove members or clients and cannot see "Private Blog." **CMSUser** - can add members, Cases (see page 26) and see "CMSUser Blog" In addition, a CMSUser can remove members, Cases (see page 26), and clients.
	MetadataDefinitionTypes (see page 3)	This enum holds the text for the three metdata definition types used by the Case Management StarterApp.

Legend

	Enumeration

Ektron.Cms.StarterApps.CaseManagement.CaseManagement.InformationArchitectureUserModes Enumeration

MembershipUser - can add members and Case to clients. They cannot remove members or clients and cannot see "Private Blog."

CMSUser - can add members, Cases (see page 26) and see "CMSUser Blog" In addition, a CMSUser can remove members, Cases (see page 26), and clients.

C#

```
public enum InformationArchitectureUserModes {
  MembershipUser,
  CMSUser
}
```

Visual Basic

```
Public Enum InformationArchitectureUserModes
  MembershipUser,
  CMSUser
End Enum
```

File

CaseManagement.vb

Members

Members	Description
MembershipUser	Can add members and Case to clients. Cannot remove members or clients. Cannot see "Private Blog."
CMSUser	Can add members, Cases (see page 26), and see "CMSUser Blog." In addition, a CMSUser can remove members, Cases (see page 26), and clients.

Remarks

This enum is used by the Public Property InformationArchitectureUserMode (see page 25)().

Ektron.Cms.StarterApps.CaseManagement.CaseManagement.MetadataDefinitionTypes Enumeration

This enum holds the text for the three metdata definition types used by the Case Management StarterApp.

C#

```
public enum MetadataDefinitionTypes {
}
```

Visual Basic

```
Public Enum MetadataDefinitionTypes
End Enum
```

File

CaseManagement.vb

Remarks

These metadata definition types are added in Initialize (see page 16)() and consumed during generation of InformationArchitecture.xml.

CaseManagement Fields

The fields of the CaseManagement class are listed here.

Private Fields

	Name	Description
	InformationAchitectureUserModeValues (see page 4)	For use with Public Property Set InformationArchitectureUserMode (see page 25)() Set "MembershipUser" as default - this is a precaution; MembershipUser cannot delete clients and Cases (see page 26), and cannot add clients.

Legend

	Data Member
	Private

CaseManagement.InformationAchitectureUserModeValues Field

C#

```
private InformationArchitectureUserModes InformationAchitectureUserModeValues =
InformationArchitectureUserModes.MembershipUser;
```

Visual Basic

```
Private InformationAchitectureUserModeValues As InformationArchitectureUserModes =
InformationArchitectureUserModes.MembershipUser
```

Description

For use with Public Property Set InformationArchitectureUserMode (see page 25)() Set "MembershipUser" as default - this is a precaution; MembershipUser cannot delete clients and Cases (see page 26), and cannot add clients.

CaseManagement Methods

The methods of the CaseManagement class are listed here.

Private Methods

	Name	Description
	AddMetadataDefinition (see page 6)	This method is called by Initialize (see page 16)() when creating the required metadata values
	GetBlogPermissionsType (see page 7)	This method gets the logical "type" of blog.
	getcasexml (see page 8)	This method collects blog component information and inserts this information into InformationArchitecture.xml. This method recursively calls itself to gather all folders/sub-folders of a particular Case.
	GetMembersXML (see page 13)	This method inserts member-information to the Client nodes in the InformationArchitecture.xml generated by GetInformationArchitectureXML (see page 11)().
	MembershipUserTraverseOnly (see page 18)	This method checks to see if a Membership user ID has been specifically granted permissions to a folder.

Legend

	Method
	Private

Public Methods

	Name	Description
	AddFolder (see page 5)	Creates StarterApps (see page 1) and Case Management folders.
	GetBreadcrumbs (see page 8)	This method gets the breadcrumb trail based on the user's navigation selection.
	GetInformationArchitecture (see page 10)	This method gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	GetInformationArchitectureXML (see page 11)	This method generates an XML-based hierarchy of clients, Cases (see page 26) and Case components to which the user has permissions.
	GetIssuesSmartFormID (see page 12)	This method returns the ID of the Milestones SmartForm.
	GetMetadataDefinitionID (see page 14)	This method gets IDs for any of the three metadata definitions for the Case Management StarterApp.

	GetSmartFormID (see page 15)	This method returns the ID of the Milestones SmartForm.
	Initialize (see page 16)	The Initialize() method is executed upon each page load to ensure StartApp has the components it needs, and returns the "Case Management" folder ID.
	RegisterMilestonesSmartForm (see page 20)	Imports the Milestones SmartFrom from "xml/MilestonesSmartForm.xml"
	RegisterTemplate (see page 21)	This method registers our templates inside CMS400 during Case Management StarterApp Initialization.
	RemoveInheritedUserGroups (see page 21)	This method removes any user groups and users assigned to a folder.
	SetFolderPermissions (see page 22)	This method breaks content and metadata inheritance for folders. You may specify which inheritance property you wish to break or specify if you wish to break both.
	SetFolderPrivate (see page 23)	This method sets the folder to private status.
	TransformInformationArchitectureXML (see page 23)	This method transforms Information Architecture XML to an XHTML unordered list.

Legend

	Method

CaseManagement.AddFolder Method

Creates StarterApps (see page 1) and Case Management folders.

C#

```
public int AddFolder(String FolderName, int ParentID);
```

Visual Basic

```
Public Function AddFolder(ByVal FolderName As String, ByVal ParentID As Integer) As Integer
```

Parameters

Parameters	Description
FolderName	This should either be "StarterApps (see page 1)" or "Case Management."
ParentID	This should either by "0" if it's the "StarterApps (see page 1)" folder or StarterAppsFolderID if it's the "Case Management" folder.

Returns

Folder ID

Remarks

This method is called by the Intialize() method.

Body Source

```
 1: Public Function AddFolder(ByVal FolderName As String, ByVal ParentID As Integer) As Integer
 2:
 3:     Dim FolderRequest As New FolderRequest
 4:     Dim FolderDataItem As New FolderData
 5:
 6:     FolderRequest.FolderName = FolderName
 7:     FolderRequest.ParentId = ParentID
 8:     FolderRequest.MetaInherited = 0
 9:     FolderRequest.StyleSheet = "NoCss.css"
10:     FolderRequest.TemplateFileName = "NoTemplate.aspx"
11:     FolderRequest.TaxonomyInherited = False
12:     FolderRequest.CategoryRequired = False
13:     FolderRequest.FolderType = Ektron.Cms.Common.EkEnumeration.FolderType.Content
14:     FolderRequest.suppressNotification = True
15:     FolderRequest.FolderDescription = "Ektron Starter Apps Folder"
16:     FolderRequest.SiteMapPathInherit = False
17:     FolderAPI.AddFolder(FolderRequest)
18:
```

```
19:     Return FolderRequest.FolderId
20: End Function
```

CaseManagement.AddMetadataDefinition Method

This method is called by Initialize (see page 16)() when creating the required metadata values

C#

```
private int AddMetadataDefinition(MetadataDefinitionTypes Type);
```

Visual Basic

```
Private Function AddMetadataDefinition(ByVal Type As MetadataDefinitionTypes) As Integer
```

Parameters

Parameters	Description
Type	Values: "cm.DefaultContent", "cm.MembershipBlog", "cm.CMSBlog"

Returns

Metadata definition ID

Remarks

- **"cm.DefaultContent"** - used to identify a content entry to load by default when the Wiki (see page 130) loads without any user-selected content.
- **"cm.MembershipBlog"** - used during the creation of InformationArchitecture.xml to identify the audience of the blog ("Membership Blogs" are actually viewable by both CMS (see page 1) users AND Membership users.)
- **"cm.CMSBlog"** - used during the creation of InformationArchitecture.xml to identify the audience of the blog ("CMS (see page 1) Blogs" are viewable only by CMS (see page 1) users.)

Body Source

```
 1: Private Function AddMetadataDefinition(ByVal Type As MetadataDefinitionTypes) As Integer
 2:     Dim Metadata As New Metadata
 3:     Dim MetadataCollection As New Ektron.Cms.ContentMetaData
 4:     Dim MetadataTypeName As String
 5:
 6:     Select Case Type
 7:         Case MetadataDefinitionTypes.wiki
 8:             MetadataTypeName = "starterapps.cm.defaultcontent"
 9:         Case MetadataDefinitionTypes.membershipblog
10:             MetadataTypeName = "starterapps.cm.membershipblog"
11:         Case MetadataDefinitionTypes.cmsblog
12:             MetadataTypeName = "starterapps.cm.cmsblog"
13:         Case Else
14:             Exit Function
15:     End Select
16:
17:     MetadataCollection.TypeName = MetadataTypeName
18:     MetadataCollection.DefaultText = "No"
19:
20:     'Title Values:
21:     'text, number, byte, double, float, integer, long, short, date, boolean, select1
(select from a list), select (multiple selections).
22:     MetadataCollection.Title = "boolean"
23:     'TagType Values:
24:     '100 - Searchable Property
25:     '1 - Meta
26:     '0 - HTML
27:     '2 - Collection
28:     '3 - List Summary
29:     '4 - Content
30:     '5 - Image
31:     '7 - File
```

```
    MetadataCollection.TagType = "100"
    MetadataCollection.Editable = True
    MetadataCollection.Separater = ";"
    MetadataCollection.SearchAllowed = True
    MetadataCollection.SelectableText = "No;Yes"
    MetadataCollection.MetaDisplayEE = False
    MetadataCollection.MetaDisplay = False
    MetadataCollection.IsSelectableOnly = True

    Try
        Metadata.AddMetaDataType(MetadataCollection)
    Catch ex As Exception
        Return -1
    End Try

    Return MetadataCollection.TypeId

End Function
```

CaseManagement.GetBlogPermissionsType Method

This method gets the logical "type" of blog.

C#

```
private String GetBlogPermissionsType(int BlogID);
```

Visual Basic

```
Private Function GetBlogPermissionsType(ByVal BlogID As Integer) As String
```

Parameters

Parameters	Description
BlogID	ID of a blog inside the Case Management StarterApp folder tree.

Returns

String: either "MembershipBlog", or "CMSBlog."

Remarks

This value is based on a metadata setting assigned at the time the blog was created. This value differentiates blogs intended for CMS (see page 1) users from those intended for CMS (see page 1) AND Membership users.

Body Source

```
Private Function GetBlogPermissionsType(ByVal BlogID As Integer) As String
    Dim Metadata As New Ektron.Cms.API.CustomFields
    Dim MetadataCollection As New Collection
    Dim MetadataType As String = "None"
    Dim BlogTypeArray(0) As String
    Dim BlogType As String

    MetadataCollection = Metadata.GetFieldsByFolder(BlogID, 1033)
    For Each MetadataItem As Collection In MetadataCollection
        If MetadataItem IsNot Nothing Then
            If MetadataItem.Item("Assigned") <> 0 Then
                MetadataType = MetadataItem.Item("CustomFieldName")
            End If
        End If
    Next
    BlogTypeArray = Split(MetadataType, ".")
    BlogType = BlogTypeArray(BlogTypeArray.Length - 1)
    Return BlogType
End Function
```

CaseManagement.GetBreadcrumbs Method

This method gets the breadcrumb trail based on the user's navigation selection.

C#

```
public String GetBreadcrumbs();
```

Visual Basic

```
Public Function GetBreadcrumbs() As String
```

Returns

An XHTML unordered list.

Remarks

To generate the breadcrumb trail, this method transforms InformationArchitecture.xml with Breadcrumbs.xslt.

Body Source

```
 1: Public Function GetBreadcrumbs() As String
 2:     Dim XMLUtils As New XMLUtilities
 3:
 4:     'Set XML Args.
 5:     XMLUtils.XMLSourceType = XMLUtilities.XMLSourceTypes.XMLDocument
 6:     XMLUtils.XMLDocument = InformationArchitectureXML
 7:     'Set XSLT Params.
 8:     XMLUtils.XSLTParams.Add("CaseID", Me.CaseID)
 9:     XMLUtils.XSLTParams.Add("Template", Me.Template)
10:     XMLUtils.XSLTParams.Add("TemplateLabel", Me.TemplateLabel)
11:     'Set XSLT Args.
12:     XMLUtils.XSLTSourceType = XMLUtilities.XSLTSourceTypes.File
13:     XMLUtils.XSLTSource = Me.IAPath & "xml/Breadcrumbs.xslt"
14:     'Transform Information Architectre.
15:     GetBreadcrumbs = XMLUtils.TransformXML()
16: End Function
```

CaseManagement.getcasexml Method

This method collects blog component information and inserts this information into InformationArchitecture.xml. This method recursively calls itself to gather all folders/sub-folders of a particular Case.

C#

```
private getcasexml(XmlNode CurrentNode, Ektron.Cms.FolderData FolderData,
System.Xml.XmlDocument IAdoc);
```

Visual Basic

```
Private Sub getcasexml(ByRef CurrentNode As XmlNode, ByRef FolderData As
Ektron.Cms.FolderData, ByRef IAdoc As System.Xml.XmlDocument)
```

Parameters

Parameters	Description
CurrentNode	The current folder in the recursive loop.
FolderData	The Ektron FolderData object of the current folder in the recursive loop.
IAdoc	The pointer to the InformationAcrichtecture.xml document being built.

Remarks

This method fills out the details of Case-level components. It collects as XML attributes the client, Case or component's...

- **label** - the name of the client, Case or component.
- **id** - the folder ID.
- **class** - enumerated as "Client," "Case," "Blog," "DiscussionBoard," "DiscussionForum," "CommunityDocuments,

"CommunityMilestones," "Content," "Domain," "Root," or "Community."

- **type** - if the node is a blog, then it also gets a "type" attribute; either "MembershipBlog", or "CMSBlog."

Body Source

```
 1: Private Sub getcasexml(ByRef CurrentNode As XmlNode, ByRef FolderData As
Ektron.Cms.FolderData, ByRef IAdoc As System.Xml.XmlDocument)
 2:     Dim Node, ErrorNode As XmlNode
 3:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 4:     Dim Permissions As New Ektron.Cms.API.Permissions
 5:     Dim XMLUtils As New XMLUtilities
 6:     Dim NodeDepth As Integer
 7:     Dim NodeDepthNavigator As System.Xml.XPath.XPathNavigator
 8:     Dim NodeClass As String = "Undetermined"
 9:     Dim GetBlogType As Boolean = False
10:     Dim BlogType As String
11:
12:     'Create a new li element to hold information regarding the current level in the Info
Architecture.
13:     Node = IAdoc.CreateElement("li")
14:
15:     'Add Title and ID attributes.
16:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "label", FolderData.Name)
17:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "id", FolderData.Id)
18:
19:     'Class the node by virtue of its depth in the hierarchy.
20:     NodeDepthNavigator = CurrentNode.CreateNavigator
21:     NodeDepth = NodeDepthNavigator.Evaluate("count(ancestor-or-self::li)")
22:     Select Case NodeDepth
23:         Case 0
24:             NodeClass = "Client"
25:         Case 1
26:             NodeClass = "Case"
27:         Case Else
28:             Select Case FolderData.FolderType
29:                 Case 0
30:                     NodeClass = "Content"
31:                 Case 1
32:                     NodeClass = "Blog"
33:                     GetBlogType = True
34:                 Case 2
35:                     NodeClass = "Domain"
36:                 Case 3
37:                     NodeClass = "DiscussionBoard"
38:                 Case 4
39:                     NodeClass = "DiscussionForum"
40:                 Case 5
41:                     NodeClass = "Root"
42:                 Case 6
43:                     Select Case FolderData.Name
44:                         Case "Documents"
45:                             NodeClass = "CommunityDocuments"
46:                         Case "Milestones"
47:                             NodeClass = "CommunityMilestones"
48:                         Case "Issues"
49:                             NodeClass = "CommunityIssues"
50:                         Case Else
51:                             NodeClass = "Community"
52:                     End Select
53:             End Select
54:     End Select
55:
56:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "class", NodeClass)
57:
58:     If GetBlogType = True Then
59:         BlogType = GetBlogPermissionsType(FolderData.Id)
```

```
60:         XMLUtils.ApplyXMLAttribute(IAdoc, Node, "type", BlogType)
61:     End If
62:
63:     'Append current folder node (with all its attributes, and child node with their
attributes) to its parent node).
64:     CurrentNode.AppendChild(Node)
65:
66:     'Get all child folders that current user has permissions to view.
67:     Dim ChildFolders As New Ektron.Cms.ContentAPI
68:     Dim ChildFolderData() As Ektron.Cms.FolderData = Nothing
69:
70:     Try
71:         ChildFolderData = ChildFolders.GetChildFoldersByFolderId(FolderData.Id)
72:         If ChildFolderData IsNot Nothing Then
73:             For Each ChildFolder As FolderData In ChildFolderData
74:                 If ContentAPI.MemberType = 1 Then
75:                     If MembershipUserTraverseOnly(ContentAPI.UserId, ChildFolder.Id) =
False Then
76:                         getcasexml(Node, ChildFolder, IAdoc)
77:                     End If
78:                 Else
79:                     getcasexml(Node, ChildFolder, IAdoc)
80:                 End If
81:             Next
82:         End If
83:     Catch ex As Exception
84:         ErrorNode = IAdoc.CreateElement("li")
85:         XMLUtils.ApplyXMLAttribute(IAdoc, ErrorNode, "status", "error")
86:         XMLUtils.ApplyXMLAttribute(IAdoc, ErrorNode, "message", ex.Message)
87:         CurrentNode.AppendChild(ErrorNode)
88:     End Try
89: End Sub
```

CaseManagement.GetInformationArchitecture Method

This method gets the information architecture (navigation/content hierarchy) the current user has permissions to access.

C#

```
public String GetInformationArchitecture();
```

Visual Basic

```
Public Function GetInformationArchitecture() As String
```

Returns

XHTML unordered list representing the clients and Cases (see page 26) with which the current user has permissions to view/interact.

Remarks

This method crawls the Case Management folder tree looking for branches to which the user has permissions to access.

Steps:

1. Represent this structure as XML.
2. Pass this XML to InformationArchitecture.xslt.
3. Return the transformed XML as XHTML unordered list.

Body Source

```
1: Public Function GetInformationArchitecture() As String
2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
3:     Dim InformationArchitectureXHTML As String = ""
4:
5:     'Get Information Architecture.
6:     If Me.InformationArchitectureXML Is Nothing Then
```

```
 7:         InformationArchitectureXML = GetInformationArchitectureXML(False)
 8:     Else
 9:         InformationArchitectureXML = Me.InformationArchitectureXML
10:     End If
11:     'Transofrm Information Architecture XML to XHTML.
12:     If (InformationArchitectureXML IsNot Nothing) Then
13:         InformationArchitectureXHTML = TransformInformationArchitectureXML()
14:     End If
15:     Return InformationArchitectureXHTML
16: End Function
```

CaseManagement.GetInformationArchitectureXML Method

This method generates an XML-based hierarchy of clients, Cases (see page 26) and Case components to which the user has permissions.

C#

```
public System.Xml.XmlDocument GetInformationArchitectureXML(Boolean IncludeMembers);
```

Visual Basic

```
Public Function GetInformationArchitectureXML(ByVal IncludeMembers As Boolean) As
System.Xml.XmlDocument
```

Returns

An XML hierarchy of clients, Cases (see page 26) and Case components.

Remarks

This hierarchy is generated once per-page load.

This hierarchy contains all information necessary to create navigation and gather context for each request.

This accomplishes three main things...

1. All permissions checks are performed once.
2. All folder and content IDs are looked-up once.
3. Context information is made available for each URI.

Body Source

```
 1: Public Function GetInformationArchitectureXML(ByVal IncludeMembers As Boolean) As
System.Xml.XmlDocument
 2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
 3:     Dim CMFolderID As Integer
 4:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 5:     Dim FolderData() As FolderData = Nothing
 6:     Dim DocumentNode, ErrorNode, CurrentNode As System.Xml.XmlNode
 7:     Dim XMLUtils As New XMLUtilities
 8:     Dim a As New Ektron.Cms.CommonApi
 9:
10:     'Initalize the IA XML document.
11:     DocumentNode = InformationArchitectureXML.CreateElement("ektron")
12:     InformationArchitectureXML.AppendChild(DocumentNode)
13:     CurrentNode = InformationArchitectureXML.DocumentElement
14:
15:     'Get the case management folder's ID.
16:     CMFolderID = Initialize()
17:
18:     'Get all the child folders of the case management folder (these are "clients").
19:     Try
20:         FolderData = ContentAPI.GetChildFoldersByFolderId(CMFolderID)
21:     Catch ex As Exception
22:         HttpContext.Current.Response.Redirect("Login.aspx", False)
23:         GetInformationArchitectureXML = Nothing
24:         FolderData = Nothing
```

```
25:         Exit Function
26:     End Try
27:
28:     If FolderData IsNot Nothing Then
29:         Try
30:             For Each Folder As FolderData In FolderData
31:                 If ContentAPI.MemberType = 1 Then
32:                     If MembershipUserTraverseOnly(ContentAPI.UserId, Folder.Id) = False
Then
33:                         getcasexml(CurrentNode, Folder, InformationArchitectureXML)
34:                     End If
35:                 Else
36:                     getcasexml(CurrentNode, Folder, InformationArchitectureXML)
37:                 End If
38:             Next
39:         Catch ex As Exception
40:             ErrorNode = InformationArchitectureXML.CreateElement("li")
41:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, ErrorNode, "status",
"error")
42:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, ErrorNode, "message",
"Could not generate information architecture.")
43:             CurrentNode.AppendChild(ErrorNode)
44:         End Try
45:     End If
46:
47:     If IncludeMembers = True Then
48:         Me.InformationArchitectureXML = InformationArchitectureXML
49:         GetMembersXML()
50:     End If
51:
52:     Return InformationArchitectureXML
53: End Function
```

CaseManagement.GetIssuesSmartFormID Method

This method returns the ID of the Milestones SmartForm.

C#

```
public int GetIssuesSmartFormID();
```

Visual Basic

```
Public Function GetIssuesSmartFormID() As Integer
```

Returns

Integer

Remarks

The Issue Tracker SmartForm is loaded by RegisterMilestonesSmartForm (see page 20)() from an external XML file.

Body Source

```
 1: Public Function GetIssuesSmartFormID() As Integer
 2:     Dim ContentAPI As New Ektron.Cms.ContentAPI()
 3:     Dim TempID As Integer
 4:     Dim SmartForms As Collection
 5:     Dim SmartForm As Collection
 6:     Dim SmartFormID As Integer = -1
 7:
 8:     'Store the UserID of the actual user in a Temp var.
 9:     TempID = ContentAPI.RequestInformationRef.CallerId
10:     'Set the UserID internal admin.
11:     ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
12:     'Get All SmartForms.
13:     SmartForms = ContentAPI.EkContentRef.GetAllXmlConfigurations("title")
14:     Try
```

```
15:         For Each SmartForm In SmartForms
16:             If (SmartForm("CollectionTitle") = "starterapps.cm.Issues") Then
17:                 SmartFormID = SmartForm("CollectionID")
18:                 Exit Try
19:             End If
20:         Next
21:     Catch ex As Exception
22:         ' handle exception however you want to.
23:     End Try
24:     'Reset users ID.
25:     ContentAPI.RequestInformationRef.CallerId = TempID
26:     Return SmartFormID
27: End Function
```

CaseManagement.GetMembersXML Method

This method inserts member-information to the Client nodes in the InformationArchitecture.xml generated by GetInformationArchitectureXML (see page 11)().

C#

```
private System.Xml.XmlDocument GetMembersXML();
```

Visual Basic

```
Private Function GetMembersXML() As System.Xml.XmlDocument
```

Returns

InformationArchitecture.xml including membership users assigned to each client.

Remarks

This method inserts a "Members" node to each client. Then, loops through the membership group assigned to the client and inserts a "Member" node (with ID, username, firstname, lastname attributes) for each client to the "Members" node.

Body Source

```
 1: Private Function GetMembersXML() As System.Xml.XmlDocument
 2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
 3:     Dim Clients As System.Xml.XmlNodeList = Nothing
 4:     Dim Users As New Ektron.Cms.API.User.User
 5:     Dim UserGroupData As New Ektron.Cms.UserGroupData
 6:     Dim Members As New Collection
 7:     Dim MembersNode, MemberNode As System.Xml.XmlNode
 8:     Dim XMLUtils As New XMLUtilities
 9:     Dim MemberData As New UserData
10:     Dim TempID As Integer
11:
12:     InformationArchitectureXML = Me.InformationArchitectureXML
13:     If (InformationArchitectureXML IsNot Nothing) Then
14:         Clients = InformationArchitectureXML.SelectNodes("//li[@class='Client']")
15:     End If
16:     If (Clients Is Nothing) Then
17:         Return InformationArchitectureXML
18:         Exit Function
19:     End If
20:     For Each Client As XmlNode In Clients
21:         Try
22:             UserGroupData = Users.GetUserGroupByName("starterapps.cm." &
Client.Attributes.GetNamedItem("label").Value)
23:         Catch ex As Exception
24:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, Client, "error",
Client.Attributes.GetNamedItem("label").Value)
25:         End Try
26:
27:         If UserGroupData IsNot Nothing Then
28:             'Create a "members" node for this client.
```

```
29:             MembersNode = InformationArchitectureXML.CreateElement("li")
30:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MembersNode, "label",
Client.Attributes.GetNamedItem("label").Value & " Members")
31:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MembersNode, "class",
"members")
32:             Client.InsertBefore(MembersNode, Client.FirstChild)
33:
34:             'Store the UserID of the actual user in a Temp var.
35:             TempID = Users.RequestInformationRef.CallerId
36:             'Set the UserID internal admin
37:             Users.RequestInformationRef.CallerId =
Ektron.Cms.Common.EkConstants.InternalAdmin
38:             'Get all users that belong to the client's membership group using Internal
Admin.
39:             Members = Users.EkUserRef.GetUsersByGroupv2_0(UserGroupData.GroupId, "")
40:
41:             If Members IsNot Nothing Then
42:                 For Each Member As Collection In Members
43:                     MemberData = Users.GetUser(Member.Item("UserID"))
44:                     MemberNode = IADoc.CreateElement("li")
45:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"class", "member")
46:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"id", Member.Item("UserID"))
47:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"username", System.Web.HttpUtility.HtmlDecode(MemberData.Username))
48:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"firstname", System.Web.HttpUtility.HtmlDecode(MemberData.FirstName))
49:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"lastname", System.Web.HttpUtility.HtmlDecode(MemberData.LastName))
50:                     MembersNode.AppendChild(MemberNode)
51:                 Next
52:             End If
53:
54:             'Reset the UserID to the ID of the actual user.
55:             Users.RequestInformationRef.CallerId = TempID
56:         End If
57:
58:     Next
59:
60:     Return InformationArchitectureXML
61: End Function
```

CaseManagement.GetMetadataDefinitionID Method

This method gets IDs for any of the three metadata definitions for the Case Management StarterApp.

C#

```
public int GetMetadataDefinitionID(MetadataDefinitionTypes Type);
```

Visual Basic

```
Public Function GetMetadataDefinitionID(ByVal Type As MetadataDefinitionTypes) As Integer
```

Parameters

Parameters	Description
Type	"Type" enumerated to restrict the lookup to the three required metadata definitions required for this application.

Returns

A single metadata definition ID.

Remarks

This method is intended for use with the Case Management StarterApp. This method is not intended to be a general purpose lookup for metadata definition IDs.

Body Source

```
 1: Public Function GetMetadataDefinitionID(ByVal Type As MetadataDefinitionTypes) As Integer
 2:     Dim Metadata As New Metadata
 3:     Dim MetadataCollection() As Ektron.Cms.ContentMetaData
 4:     Dim MetadataType As New Ektron.Cms.ContentMetaData
 5:     Dim SelectedMetadataType As String
 6:
 7:     Select Case Type
 8:         Case MetadataDefinitionTypes.wiki
 9:             SelectedMetadataType = "starterapps.cm.defaultcontent"
10:         Case MetadataDefinitionTypes.cmsblog
11:             SelectedMetadataType = "starterapps.cm.cmsblog"
12:         Case MetadataDefinitionTypes.membershipblog
13:             SelectedMetadataType = "starterapps.cm.membershipblog"
14:         Case Else
15:             SelectedMetadataType = "None"
16:     End Select
17:
18:     MetadataCollection = Metadata.GetMetaDataTypes("Title")
19:     For Each MetadataType In MetadataCollection
20:         If MetadataType.TypeName.ToLower() = SelectedMetadataType.ToLower() Then
21:             GetMetadataDefinitionID = MetadataType.TypeId
22:             Exit For
23:         Else
24:             GetMetadataDefinitionID = -1
25:         End If
26:     Next
27:
28:     Return GetMetadataDefinitionID
29: End Function
```

CaseManagement.GetSmartFormID Method

This method returns the ID of the Milestones SmartForm.

C#

```
public int GetSmartFormID();
```

Visual Basic

```
Public Function GetSmartFormID() As Integer
```

Returns

Integer

Remarks

The Milestones SmartForm is loaded by RegisterMilestonesSmartForm (see page 20)() from an external XML file.

Body Source

```
 1: Public Function GetSmartFormID() As Integer
 2:     Dim ContentAPI As New Ektron.Cms.ContentAPI()
 3:     Dim TempID As Integer
 4:     Dim SmartForms As Collection
 5:     Dim SmartForm As Collection
 6:     Dim SmartFormID As Integer = -1
 7:
 8:     'Store the UserID of the actual user in a Temp var.
 9:     TempID = ContentAPI.RequestInformationRef.CallerId
10:     'Set the UserID internal admin.
11:     ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
12:     'Get All SmartForms.
13:     SmartForms = ContentAPI.EkContentRef.GetAllXmlConfigurations("title")
14:     Try
```

```
15:         For Each SmartForm In SmartForms
16:             If (SmartForm("CollectionTitle") = "starterapps.cm.Milestones") Then
17:                 SmartFormID = SmartForm("CollectionID")
18:                 Exit Try
19:             End If
20:         Next
21:     Catch ex As Exception
22:         ' handle exception however you want to.
23:     End Try
24:     'Reset users ID.
25:     ContentAPI.RequestInformationRef.CallerId = TempID
26:     Return SmartFormID
27: End Function
```

CaseManagement.Initialize Method

The Initialize() method is executed upon each page load to ensure StartApp has the components it needs, and returns the "Case Management" folder ID.

C#

```
public int Initialize();
```

Visual Basic

```
Public Function Initialize() As Integer
```

Returns

Case Management folder ID.

Remarks

This method...

1. checks to see if the "StarterApps (see page 1)" and "Case Management" folders exists, and if not, creates them.
2. checks to see if the "starterapps.cm" CMS (see page 1) user group exists, and if not, creates it.
3. checks to see if the three required metadata values "cm.DefaultContent", "cm.Blog", and "cm.PrivateBlog" exist and if not, creates them.
4. checks to see if the required .aspx templates are registered with CMS400 and if not, registers them.

Body Source

```
 1: Public Function Initialize() As Integer
 2:     Dim FolderData() As FolderData
 3:     Dim StarterAppsFolderID As Integer = -1
 4:     Dim CMFolderID As Integer = -1
 5:     Dim CMSGroupID As Integer = -1
 6:     Dim StarterAppsFolderExists As Boolean = False
 7:     Dim CMFolderExists As Boolean = False
 8:     Dim GroupName As String = ""
 9:     Dim CMUser As New StarterApps.CaseManagement.Users
10:     Dim Metadata As New Metadata
11:     Dim MetadataCollection() As Ektron.Cms.ContentMetaData
12:     Dim MetadataType As New Ektron.Cms.ContentMetaData
13:     Dim MetadataID, SmartFormID As Integer
14:     Dim UserGroup As New API.User.User
15:     Dim MembershipBlogMetadataExists, CMSBlogMetadataExists, WikiMetadataExists As Boolean
16:
17:     'Get all child folders of the CMS Root folder and see if the
18:     'folder "Starter Apps" has already been created.
19:     'If so, set StarterAppsFolderExists to TRUE
20:     FolderData = FolderAPI.GetChildFolders(0, False)
21:     If Not FolderData Is Nothing Then
22:         For Each FolderDataItem As FolderData In FolderData
23:             If FolderDataItem.Name = "Starter Apps" Then
24:                 StarterAppsFolderID = FolderDataItem.Id
```

```
25:                 StarterAppsFolderExists = True
26:                 Exit For
27:             End If
28:         Next
29:     End If
30:
31:     Try
32:         'If the "Application Accelerators" folder doesn't exist, create it
33:         If StarterAppsFolderExists = False Then
34:             StarterAppsFolderID = AddFolder("Starter Apps", 0)
35:             'Break inheritance and remove inherited permissions
36:             SetFolderPermissions(StarterAppsFolderID)
37:             'Remove permissions for all inherited groups
38:             RemoveInheritedUserGroups(StarterAppsFolderID)
39:             'Give everyone traverse-only permission on this folder
40:             CMUser.AddUserGroupToFolder(StarterAppsFolderID, EVERYONEGROUP, True)
41:             'Add CMS Case management user group and add to "Starter Apps" folder
42:             GroupName = "starterapps.cm"
43:             CMSGroupID = CMUser.AddCMSUserGroup(GroupName)
44:             CMUser.AddUserGroupToFolder(StarterAppsFolderID, CMSGroupID, True)
45:             'Make Folder is marked 'Private'
46:             SetFolderPrivate(StarterAppsFolderID)
47:         End If
48:     Catch ex As Exception
49:         'clean up
50:         StarterAppsFolderID = FolderAPI.GetFolderId("Starter Apps", 0)
51:         If (StarterAppsFolderID <> -1) Then
52:             FolderAPI.DeleteFolderById(StarterAppsFolderID)
53:         End If
54:         If (CMSGroupID <> -1) Then
55:             UserGroup.DeleteUserGroup(CMSGroupID)
56:         End If
57:         Throw ex
58:         Exit Function
59:     End Try
60:     If (StarterAppsFolderID > 0) Then
61:         'Get all child folders of the "Application Accelerators" folder
62:         'to see if the "Case Management" foldler exists.
63:         FolderData = FolderAPI.GetChildFolders(StarterAppsFolderID, False)
64:         If Not FolderData Is Nothing Then
65:             For Each FolderDataItem As FolderData In FolderData
66:                 If FolderDataItem.Name = "Case Management" Then
67:                     CMFolderID = FolderDataItem.Id
68:                     CMFolderExists = True
69:                     Exit For
70:                 End If
71:             Next
72:         End If
73:
74:
75:         'If the cm folder doesn't exist, folder doesn't exist, create it.
76:         If CMFolderExists = False Then
77:             CMFolderID = AddFolder("Case Management", StarterAppsFolderID)
78:             'Break inheritance and remove inherited permissions.
79:             SetFolderPermissions(CMFolderID)
80:             'Remove permissions for all inherited groups.
81:             RemoveInheritedUserGroups(CMFolderID)
82:             'Add CMS user group and add to CM folder.
83:             GroupName = "starterapps.cm"
84:             CMSGroupID = CMUser.AddCMSUserGroup(GroupName)
85:             CMUser.AddUserGroupToFolder(CMFolderID, CMSGroupID, False)
86:             'Make Folder is marked 'Private'.
87:             SetFolderPrivate(CMFolderID)
88:         End If
89:
90:         'See if the metadata mefinition for default content exists.
```

```
        'If not, create it.

        MembershipBlogMetadataExists = False
        CMSBlogMetadataExists = False
        WikiMetadataExists = False

        MetadataCollection = Metadata.GetMetaDataTypes("Title")
        For Each MetadataType In MetadataCollection
            Select Case MetadataType.TypeName.ToLower()
                Case "starterapps.cm.defaultcontent"
                    MetadataID = MetadataType.TypeId
                    WikiMetadataExists = True
                Case "starterapps.cm.membershipblog"
                    MetadataID = MetadataType.TypeId
                    MembershipBlogMetadataExists = True
                Case "starterapps.cm.cmsblog"
                    MetadataID = MetadataType.TypeId
                    CMSBlogMetadataExists = True
            End Select
        Next

        If WikiMetadataExists = False Then
            MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.Wiki)
        End If

        If MembershipBlogMetadataExists = False Then
            MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.MembershipBlog)
        End If

        If CMSBlogMetadataExists = False Then
            MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.CMSBlog)
        End If

        'Register Page Templates.
        RegisterTemplate("Cases.aspx")
        RegisterTemplate("Wiki.aspx")
        RegisterTemplate("Dynamic.aspx")
        RegisterTemplate("Discussion.aspx")
        RegisterTemplate("Documents.aspx")
        RegisterTemplate("Milestones.aspx")
        RegisterTemplate("Blog.aspx")
        RegisterTemplate("Login.aspx")
        RegisterTemplate("Issues.aspx")

        'Import Milestones Smartform.
        SmartFormID = GetSmartFormID()
        If SmartFormID = -1 Then
            RegisterMilestonesSmartForm()
        End If

        'Import Issues Smartform.
        SmartFormID = GetIssuesSmartFormID()
        If SmartFormID = -1 Then
            RegisterIssuesSmartForm()
        End If
    End If
    Return CMFolderID
End Function
```

CaseManagement.MembershipUserTraverseOnly Method

This method checks to see if a Membership user ID has been specifically granted permissions to a folder.

C#

```
private Boolean MembershipUserTraverseOnly(int UserId, int FolderID);
```

Visual Basic

```
Private Function MembershipUserTraverseOnly(ByVal UserId As Integer, ByVal FolderID As
Integer) As Boolean
```

Parameters

Parameters	Description
UserId	The Membership user's ID.
FolderID	The folder ID of the folder we wish to check.

Returns

True - if the user can only Traverse the folder.

False - if the user has been granted permissions to interact (e.g. add, delete, edit...) with the content in the folder.

Remarks

By default, Membership users have Traverse permissions for all folders. This is a hardcoded setting! This method figures out if a Membership user has been specifically granted permission for a folder via direct assignment of the user to the folder, or via a Membership user group that has been assigned to the folder. If the user has permissions beyond Traverse, the return value is set to "False", else it is set to "True."

Body Source

```
 1: Private Function MembershipUserTraverseOnly(ByVal UserId As Integer, ByVal FolderID As
Integer) As Boolean
 2:     Dim UserAPI As New Ektron.Cms.API.User.User
 3:     Dim GroupData() As Ektron.Cms.GroupData
 4:     Dim PermissionsAPI As New Ektron.Cms.API.Permissions
 5:     Dim PermissionsData() As Ektron.Cms.UserPermissionData
 6:
 7:     'Get all the groups to which the user is assigned.
 8:     GroupData = UserAPI.GetGroupsUserIsIn(UserId, "groupname")
 9:     'Get the permissions data for the folder being examined.
10:     PermissionsData = PermissionsAPI.GetUserPermissions(FolderID, "folder", 0, "")
11:     'Set the default return value - that the Membership user has ONLY traverse permissions.
12:     MembershipUserTraverseOnly = True
13:
14:     For Each PermissionsItem As UserPermissionData In PermissionsData
15:         If PermissionsItem IsNot Nothing Then
16:             'Check to see if the user has been granted access to the folder directly.
17:             If PermissionsItem.UserId = UserId Then
18:                 'In this case, the user has specifically been granted access to the folder.
19:                 'Set return value to false - user has been granted more access
20:                 'than the default traverse-only permission.
21:                 MembershipUserTraverseOnly = False
22:                 Exit For
23:             End If
24:             'Check to see if the user has been granted access to the folder via a
Membership group.
25:             For Each UserGroup As GroupData In GroupData
26:                 If UserGroup.GroupId = PermissionsItem.GroupId Then
27:                     'In this case, the user belongs to a Membership user group that has
28:                     'specifically been granted access to the folder.
29:                     'Set return value to false - user has been granted more access
30:                     'than the default traverse-only permission.
31:                     MembershipUserTraverseOnly = False
32:                     Exit For
33:                 End If
34:             Next
35:         End If
36:         If MembershipUserTraverseOnly = False Then
37:             Exit For
38:         End If
```

```
39:     Next
40: End Function
```

CaseManagement.RegisterMilestonesSmartForm Method

Imports the Milestones SmartFrom from "xml/MilestonesSmartForm.xml"

C#

```
public RegisterMilestonesSmartForm();
```

Visual Basic

```
Public Sub RegisterMilestonesSmartForm()
```

Remarks

You can later get the ID of this SmartForm via GetSmartFormID (see page 15)().

Body Source

```
 1: Public Sub RegisterMilestonesSmartForm()
 2:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 3:     Dim TempID As Integer
 4:     Dim SmartFormData As New Collection
 5:     Dim SmartFormXML, SmartFormDisplayXSLT As String
 6:     Dim XMLDoc, XSLDoc As New System.Xml.XmlDocument
 7:     Dim SmartFormID As Integer
 8:     Dim EkContent As New Ektron.Cms.Content.EkContent
 9:     Dim Package As New Collection()
10:
11:     'Store the current users's ID in a temprorary integer var.
12:     TempID = ContentAPI.RequestInformationRef.CallerId
13:     'Set the current user's ID to the INTERNALADMIN constant.
14:     ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
15:
16:     'Get Milestones SmartForm Package XML.
17:
XMLDoc.Load(System.Web.HttpContext.Current.Server.MapPath("xml/MilestonesSmartForm.xml"))
18:     SmartFormXML = XMLDoc.InnerXml
19:     'Get Milestones SmartForm Display XSLT.
20:
XSLDoc.Load(System.Web.HttpContext.Current.Server.MapPath("xml/MilestonesSmartForm.xsl"))
21:     SmartFormDisplayXSLT = XSLDoc.InnerXml
22:
23:     'Below are the Collection Paramters for the SmartForm Collection.
24:     SmartFormData.Add("0", "DefaultXSLT")
25:     SmartFormData.Add("starterapps.cm.Milestones", "CollectionTitle")
26:     SmartFormData.Add("This smart form is associated with Milestones in the Case
Management StarterApp", "CollectionDescription")
27:     SmartFormData.Add(System.Web.HttpContext.Current.Server.MapPath("xml"), "PhysicalPath")
28:     SmartFormData.Add("", "displayXslt")
29:     SmartFormData.Add("", "EditXSLT")
30:     SmartFormData.Add("", "SaveXSLT")
31:     SmartFormData.Add("", "XSLT1")
32:     SmartFormData.Add("", "XSLT2")
33:     SmartFormData.Add("", "XSLT3")
34:     SmartFormData.Add("", "XSLT4")
35:     SmartFormData.Add("", "XSLT5")
36:     SmartFormData.Add("", "XmlSchema")
37:     SmartFormData.Add("", "XmlNameSpace")
38:     SmartFormData.Add("", "XmlAdvConfig")
39:     SmartFormID = ContentAPI.AddXmlConfiguration(SmartFormData)
40:
41:     'Add the SmartForm Package.
42:     Package.Add(SmartFormID, "XmlCollectionID")
43:     Package.Add(SmartFormXML, "Package")
44:     Package.Add(SmartFormDisplayXSLT, "displayXslt")
```

```
45:     Package.Add("", "DesignStylesheet")
46:     ContentAPI.UpdatexmlCollectionPackage(Package)
47:
48:     'Set the current user's ID back.
49:     ContentAPI.RequestInformationRef.CallerId = TempID
50: End Sub
```

CaseManagement.RegisterTemplate Method

This method registers our templates inside CMS400 during Case Management StarterApp Initialization.

C#

```
public RegisterTemplate(String TemplateName);
```

Visual Basic

```
Public Sub RegisterTemplate(ByVal TemplateName As String)
```

Parameters

Parameters	Description
TemplateName	The name of the template to resister.

Remarks

Path to the templates is hard coded in this method as "StarterApps (see page 1)/CaseManagement (see page 2)/"

Body Source

```
 1: Public Sub RegisterTemplate(ByVal TemplateName As String)
 2:     Dim TemplateData As New Collection()
 3:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 4:     Dim ExistingTemplates As TemplateData() = ContentAPI.GetAllTemplates("")
 5:     Dim TemplateFileName As String
 6:     Dim TemplateExists As Boolean = False
 7:
 8:     TemplateFileName = "StarterApps/CaseManagement/" & TemplateName
 9:
10:     'Check to see if template already exists
11:     For Each Template As TemplateData In ExistingTemplates
12:         If Template.FileName = TemplateFileName Then
13:             TemplateExists = True
14:             Exit For
15:         End If
16:     Next
17:
18:     'If the template doesn't exist, add it
19:     If TemplateExists = False Then
20:         TemplateData.Add(TemplateFileName, "TemplateFileName")
21:         ContentAPI.EkContentRef.AddTemplatev2_0(TemplateData)
22:     End If
23: End Sub
```

CaseManagement.RemoveInheritedUserGroups Method

This method removes any user groups and users assigned to a folder.

C#

```
public RemoveInheritedUserGroups(int FolderID);
```

Visual Basic

```
Public Sub RemoveInheritedUserGroups(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	The ID of the folder from which to remove all user groups.

Remarks

This method is used to "clean up" a folder after breaking content inheritance and before adding new user groups to create a new permissions model.

Notes

This method removes both all users and all user groups.

Body Source

```
 1: Public Sub RemoveInheritedUserGroups(ByVal FolderID As Integer)
 2:     Dim Permissions As New Ektron.Cms.API.Permissions
 3:     Dim PermissionsData() As Ektron.Cms.UserPermissionData
 4:
 5:     PermissionsData = Permissions.GetUserPermissions(FolderID, "folder", 0, "")
 6:
 7:     If PermissionsData IsNot Nothing Then
 8:         For Each UserGroup As UserPermissionData In PermissionsData
 9:             Select Case UserGroup.GroupId
10:                 Case -1
11:                     If UserGroup.UserId > 1 Then
12:                         'This is a user - delete it.
13:                         Permissions.DeleteItemPermission(UserGroup.UserId, False,
FolderID, EkEnumeration.CMSObjectTypes.Folder)
14:                     End If
15:                 Case 1
16:                     'This is the admin group - do nothing.
17:                 Case Else
18:                     'This is a group - delete it.
19:                     Permissions.DeleteItemPermission(UserGroup.GroupId, True, FolderID,
EkEnumeration.CMSObjectTypes.Folder)
20:             End Select
21:         Next
22:     End If
23: End Sub
```

CaseManagement.SetFolderPermissions Method

This method breaks content and metadata inheritance for folders. You may specify which inheritance property you wish to break or specify if you wish to break both.

C#

```
public SetFolderPermissions(int FolderID);
```

Visual Basic

```
Public Sub SetFolderPermissions(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	Folder ID of the folder you wish to break inheritance.

Remarks

Breaking folder inheritance allows us to apply the three required Case Management StarterApp metadata types only to the folders that require them. Only Blogs should be assigned the "blogs" metadata types. The "Case/wiki (see page 130)" folder should only be assigned the "default content" metadata type

Body Source

```
1: Public Sub SetFolderPermissions(ByVal FolderID As Integer)
2:     Dim PermissionData As New Ektron.Cms.UserPermissionData
3:     Dim Permissions As New Ektron.Cms.API.Permissions
4:
5:     'Break Folder Inheritance
```

```
6:     Permissions.DisableFolderInheritance(FolderID)
7: End Sub
```

CaseManagement.SetFolderPrivate Method

This method sets the folder to private status.

C#

```
public SetFolderPrivate(int FolderID);
```

Visual Basic

```
Public Sub SetFolderPrivate(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	Folder ID of the folder to make private.

Remarks

This is a workaround since there is no direct API call to ensure a folder is set to private. To ensure this method executes properly, we must borrow the internal admin's rights by switching from the user's ID who is setting the folder to private to the constant INTERNALADMIN. Then, switch it back before finishing execution. Calling this method without setting the internal admin, even with an authorized CMS (see page 1) user, results in throwing an error. Since this user has already been authenticated to perform this action (creating a client or Case folder), we can use internal admin to avoid this problem.

Body Source

```
 1: Public Sub SetFolderPrivate(ByVal FolderID As Integer)
 2:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 3:     Dim EkContent As Ektron.Cms.Content.EkContent = ContentAPI.EkContentRef
 4:     Dim PageData As New Collection
 5:     Dim TempID As Integer
 6:
 7:     'Store the current users's ID in a temprorary integer var.
 8:     TempID = ContentAPI.RequestInformationRef.CallerId
 9:     'Set the current user's ID to the INTERNALADMIN constant.
10:     ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
11:     PageData.Add(FolderID, "ItemID")
12:     PageData.Add("folder", "RequestType")
13:
14:     'EkContent.DisableItemPrivateSettingv2_0(pagedata) ' to make public
15:     EkContent.EnableItemPrivateSettingv2_0(PageData) ' to make private
16:
17:     'Set the current user's ID back
18:     ContentAPI.RequestInformationRef.CallerId = TempID
19: End Sub
```

CaseManagement.TransformInformationArchitectureXML Method

This method transforms Information Architecture XML to an XHTML unordered list.

C#

```
public String TransformInformationArchitectureXML();
```

Visual Basic

```
Public Function TransformInformationArchitectureXML() As String
```

Returns

An XHTML unordered list.

Remarks

This method is used to create the client/Case navigation tree that the current user has permissions to traverse.

Body Source

```
 1: Public Function TransformInformationArchitectureXML() As String
 2:     Dim XMLUtils As New XMLUtilities
 3:
 4:     'Set XML Args.
 5:     XMLUtils.XMLSourceType = XMLUtilities.XMLSourceTypes.XMLDocument
 6:     XMLUtils.XMLDocument = InformationArchitectureXML
 7:     'Set XSLT Params.
 8:     XMLUtils.XSLTParams.Add("UserMode", Me.InformationArchitectureUserMode)
 9:     XMLUtils.XSLTParams.Add("ClientID", Me.ClientID)
10:     XMLUtils.XSLTParams.Add("CaseID", Me.CaseID)
11:     'Set XSLT Args.
12:     XMLUtils.XSLTSourceType = XMLUtilities.XSLTSourceTypes.File
13:     XMLUtils.XSLTSource = Me.IAPath & "xml/InformationArchitecture.xslt"
14:     'Transform Information Architectre.
15:     TransformInformationArchitectureXML = XMLUtils.TransformXML()
16: End Function
```

CaseManagement Properties

The properties of the CaseManagement class are listed here.

Public Properties

	Name	Description
	CaseID (see page 24)	The CaseID property is consumed by several XMLXSLT transformations.
	ClientID (see page 25)	The ClientID property is the FolderID of the selected client and is consumed by several methods in the CaseManagement (see page 2) namespace.
	IAPath (see page 25)	The IAPath Property indicates the location of InformationArchitecture.xslt.
	InformationArchitectureUserMode (see page 25)	This Property is used to determine "action" rights for users and is consumed by InformationArchitecture.xslt when called by the "TransformInformationArchitetureXML()." Default value is "MembershipUser."
	InformationArchitectureXML (see page 25)	The InformationArchitectureXML property is used to hold the the information architecture prior to being transformed with XSLT.
	Template (see page 26)	The Template property is used in several XSLT transformations to identify the aspx template associated with a link.
	TemplateLabel (see page 26)	The TemplateLabel property indicates the text to display for an aspx template associated with a particular link. This property is consumed by several XSLT transformations.

Legend

	Property

CaseManagement.CaseID Property

The CaseID property is consumed by several XMLXSLT transformations.

C#

```
public int CaseID;
```

Visual Basic

```
Public Property CaseID() As Integer
```

Returns

FolderID of the selected Case.

Remarks

There is an application-level variable, `Ektron_StarterApps_CaseManagement_CaseID_~integer~`, that indicates the current Case. This is set in Main.master.vb. This variable ensures that the current Case's folder ID is always available; even if it's not part of the querystring.

CaseManagement.ClientID Property

The ClientID property is the FolderID of the selected client and is consumed by several methods in the CaseManagement (see page 2) namespace.

C#

```
public int ClientID;
```

Visual Basic

```
Public Property ClientID() As Integer
```

Returns

FolderID of the selected client.

Remarks

This is used all over the place.

CaseManagement.IAPath Property

The IAPath Property indicates the location of InformationArchitecture.xslt.

C#

```
public String IAPath;
```

Visual Basic

```
Public Property IAPath() As String
```

Returns

Relative location path - e.g. "../"

Description

String

Remarks

The Information Architecture XSLT is called from several physical locations; site root, and from AJAX components under "/actions." This property allows a method calling this XSLT to identify its relative location to InformationArchitecture.xslt.

CaseManagement.InformationArchitectureUserMode Property

This Property is used to determine "action" rights for users and is consumed by InformationArchitecture.xslt when called by the "TransformInformationArchitetureXML()." Default value is "MembershipUser."

C#

```
public InformationArchitectureUserModes InformationArchitectureUserMode;
```

Visual Basic

```
Public Property InformationArchitectureUserMode() As InformationArchitectureUserModes
```

CaseManagement.InformationArchitectureXML Property

The InformationArchitectureXML property is used to hold the the information architecture prior to being transformed with XSLT.

C#

```
public System.Xml.XmlDocument InformationArchitectureXML;
```

Visual Basic

```
Public Property InformationArchitectureXML() As System.Xml.XmlDocument
```

Returns

The Information Architecture XmlDocument.

Description

XmlDocument

Remarks

IA XML is transformned in several places - to generate the client list, to get the breadcrumbs, to fill out links in the Case navigation widget.

CaseManagement.Template Property

The Template property is used in several XSLT transformations to identify the aspx template associated with a link.

C#

```
public String Template;
```

Visual Basic

```
Public Property Template() As String
```

Returns

The ASPX template associated with a link.

Description

String

Remarks

This is the template target of a link. See property TemplateLabel (see page 26)().

CaseManagement.TemplateLabel Property

The TemplateLabel property indicates the text to display for an aspx template associated with a particular link. This property is consumed by several XSLT transformations.

C#

```
public String TemplateLabel;
```

Visual Basic

```
Public Property TemplateLabel() As String
```

Returns

The display string for a link template.

Description

String

Remarks

This label is what shows up to the user.

Cases Class

Contains methods and properties that manage the Case aspect of the Case management site. This includes functionality for Blogs, Discussion Boards, Taxonomy, Wiki (see page 130) and Folders.

Class Hierarchy

Ektron.Cms.StarterApps.CaseManagement.Cases

C#

```
public class Cases;
```

Visual Basic

```
Public Class Cases
```

File

CaseManagement.vb

Cases Methods

The methods of the Cases class are listed here.

Private Methods

	Name	Description
	DefaultBlogContentText (see page 38)	Returns default Blog content.
	DefaultWikiContentText (see page 38)	Returns default Wiki (see page 130) content.
	GetDefaultTaxonomy (see page 40)	This method loads the default taxonomy ("DefaultCaseTaxonomy.xml") into a string.
	LoremIpsumLong (see page 40)	This method generates placeholder text.
	LoremIpsumShort (see page 41)	This method generates placeholder text.
	MakeFolderNameUnique (see page 42)	This method ensures a Case name is unique.
	MakeFolderReadOnly (see page 42)	This method is for use with the Milestones folder in the Case Management Starter App.

Legend

	Method
	Private

Protected Methods

	Name	Description
	AddBlog (see page 28)	This method creates a blog for the Case Management StarterApp.
	AddDefaultBlogEntry (see page 35)	This method creates a default blog entry as a placeholder (Lorem Ipsum text).
	AddDiscussion (see page 35)	This method adds a discussion to the Case Management StarterApp.
	AddFolder (see page 36)	This method adds a CMS (see page 1) folder to the Case Management StarterApp.

Legend

	Method
	Protected

Public Methods

	Name	Description
	AddCase (see page 29)	This method adds a Case to the Case Management StarterApp.
	AddCMCommunityFolder (see page 34)	This method adds a Community folder to the Case Management StarterApp.
	AddMetadataDefinitionToFolder (see page 37)	This method assigns a metadata definition (by ID) to a folder (by ID).

	GetAllCases (see page 38)	This method retrieves all Case folder names and IDs that the user has permissions to view.
	GetCaseTaxonomyID (see page 39)	Gets the ID of the taxonomy associated with the Case folder.
	RemoveCase (see page 43)	This method removes a Case from the Case Management StarterAPP.
	SetFolderTemplate (see page 44)	This method sets the default .aspx template for a folder.

Legend

	Method

Cases.AddBlog Method

This method creates a blog for the Case Management StarterApp.

C#

```
protected int AddBlog(int ParentFolderID, String BlogName, String BlogTitle, String
BlogDescription);
```

Visual Basic

```
Protected Function AddBlog(ByVal ParentFolderID As Integer, ByVal BlogName As String, ByVal
BlogTitle As String, ByVal BlogDescription As String) As Integer
```

Parameters

Parameters	Description
ParentFolderID	The ID of the folder that will become the parent to the blog to be added.
BlogName	The name of the blog.
BlogTitle	The title of the blog.
BlogDescription	The description of the blog. (This can be empty string "")

Returns

The ID of the newly created blog.

Remarks

Some details regarding how blogs are being created in this method...

1. Blog Visibility is set to private.
2. EnableComments is set to True.
3. ModerateComments is set to False.
4. CommentsRequireAuthentication is set to True.
5. No categories are added.
6. No blogroll information is added.

Body Source

```
 1: Protected Function AddBlog(ByVal ParentFolderID As Integer, ByVal BlogName As String,
ByVal BlogTitle As String, ByVal BlogDescription As String) As Integer
 2:     Dim Name, Title, Description As String
 3:     Dim Categories(1) As String
 4:     Dim Blogroll As New Ektron.Cms.BlogRoll 'This is empty.
 5:     Dim EnableComments, ModerateComments, CommentsRequireAuthentication As Boolean
 6:     Dim BlogID As Integer
 7:     Dim NewBlog As New Ektron.Cms.API.Content.Blog
 8:
 9:
10:     Name = BlogName
11:     Title = BlogTitle
12:     Description = BlogDescription
13:     EnableComments = True
14:     ModerateComments = False
```

```
15:     CommentsRequireAuthentication = True
16:     Categories(0) = "Status"
17:     Categories(1) = "Questions"
18:
19:     BlogID = NewBlog.AddBlog(ParentFolderID, Name, Title, Description,
EkEnumeration.BlogVisibility.Private, EnableComments, ModerateComments,
CommentsRequireAuthentication, Categories, Blogroll)
20:
21:     Return BlogID
22: End Function
```

Cases.AddCase Method

This method adds a Case to the Case Management StarterApp.

C#

```
public AddCase();
```

Visual Basic

```
Public Sub AddCase()
```

Remarks

This method does several things...

1. Creates the Case folder (this is the Wiki (see page 130) folder) as a Community Folder and associates this folder with "Wiki.aspx."
2. Breaks inheritance and sets the Case folder to PRIVATE.
3. Creates a default Wiki (see page 130) Case entry (a few paragraphs of Lorem Ipsum).
4. Imports the default cm Taxonomy for use with the Wiki (see page 130).
5. Creates the Documents folder as Community Folder and associates this folder with "Documents.aspx."
6. Breaks inheritance and sets the Documents folder to PRIVATE.
7. Creates the Milestones folder as Community Folder and associates this folder with "Milestones.aspx."
8. Breaks inheritance and sets the Milestones folder to PRIVATE.
9. Creates the MembershipBlog and associates this folder with "Blog.aspx."
10. Breaks inheritance and sets permissions for this folder.
11. Creates the CMSBlog and associates this folder with "Blog.aspx."
12. Breaks inheritance and sets permissions for this folder.
13. Creates the Discussion folder and associates this folder with "Discussion.aspx."
14. Breaks inheritance and sets permissions for this folder.

Body Source

```
 1: Public Sub AddCase()
 2:
 3:     'Ektron.CaseManagement Namespace
 4:     Dim CM As New Ektron.Cms.StarterApps.CaseManagement.CaseManagement
 5:     Dim Clients As New Clients
 6:     Dim Users As New Users
 7:
 8:     'Ektron
 9:     Dim NewCaseFolder As New FolderRequest
10:     Dim FolderRequest As New Ektron.Cms.FolderRequest
11:     Dim FolderData As New Ektron.Cms.FolderData
12:     Dim FolderAPI As New Ektron.Cms.API.Folder
13:     Dim ClientGroup As New Ektron.Cms.API.User.User
14:     Dim ContentAPI As New Ektron.Cms.ContentAPI
15:     Dim Taxonomy As New Ektron.Cms.API.Content.Taxonomy
```

```
 16:     Dim TaxonomyBaseDatum As New Ektron.Cms.TaxonomyBaseData
 17:     Dim TaxonomySyncRequest As New Ektron.Cms.TaxonomySyncRequest
 18:     Dim TaxonomyContentRequest As New TaxonomyContentRequest
 19:     Dim TaxonomyRequest As New Ektron.Cms.TaxonomyRequest
 20:     Dim UserGroup As New API.User.User
 21:     Dim UserGroupData As New UserGroupData
 22:     Dim Permissions As New Ektron.Cms.API.Permissions
 23:
 24:     'General
 25:     Dim TaxonomyID, TaxonomyCategoryID As Integer
 26:     Dim CMFolderID, ClientFolderID, CaseFolderID, DocumentsFolderID, IssuesFolderID,
MilestonesFolderID, CMSGroupID, BlogID, DiscussionID, MetadataDefinitionID, IssuesSmartFormID,
MilestonesSmartFormID, DefaultWikiContentID As Integer
 27:
 28:     Dim DefaultTaxonomy, TaxonomyPath As String
 29:
 30:     Dim SmartFormCollection, BlogPermissionData As New Collection()
 31:
 32:     'Get CM folder ID.
 33:     CMFolderID = CM.Initialize
 34:     CM.Initialize()
 35:     'Get Client folder ID.
 36:     ClientFolderID = Me.ClientID
 37:     'Set ClientName property.
 38:     FolderData = FolderAPI.GetFolder(ClientFolderID)
 39:     Me.ClientName = FolderData.Name
 40:     'Get CMS group ID.
 41:     CMSGroupID = Users.AddCMSUserGroup("starterapps.cm")
 42:
 43:     '- Case Folder -
 44:     '-------------------------------------------------------------------------------
 45:     'Get metadta definition ID for default content for wiki (so we can assign a "default"
content block).
 46:     MetadataDefinitionID =
cm.GetMetadataDefinitionID(CaseManagement.MetadataDefinitionTypes.Wiki)
 47:
 48:     'Add Case folder.
 49:     Me.CaseName = MakeFolderNameUnique(ClientFolderID, Me.CaseName)
 50:     CaseFolderID = AddCMCommunityFolder(ClientFolderID, Me.CaseName, "Case Folder",
MetadataDefinitionID)
 51:     'Break Inheritance and set Permissions for Case folder.
 52:     cm.SetFolderPermissions(CaseFolderID)
 53:     'Make Folder is marked 'Private'.
 54:     cm.SetFolderPrivate(CaseFolderID)
 55:
 56:     'Set folder template.
 57:     SetFolderTemplate(CaseFolderID, "dynamic.aspx")
 58:
 59:     'Get the default taxonomy and import it.
 60:     Try
 61:         DefaultTaxonomy = GetDefaultTaxonomy()
 62:         TaxonomyID = Taxonomy.ImportTaxonomy(DefaultTaxonomy, "cm." & Me.ClientName & "."
& Me.CaseName)
 63:     Catch ex As Exception
 64:         RemoveCase(CaseFolderID)
 65:         Exit Sub
 66:     End Try
 67:
 68:     ''Add the Case (Wiki) folder  to the root of the Taxonomy so that all Wiki content
gets indexed.
 69:     'TaxonomySyncRequest.SyncIdList = CaseFolderID
 70:     'TaxonomySyncRequest.TaxonomyId = TaxonomyID
 71:     'TaxonomySyncRequest.TaxonomyLanguage = ContentAPI.DefaultContentLanguage
 72:     'Taxonomy.AddTaxonomySyncFolder(TaxonomySyncRequest)
 73:
 74:     ' Add the Taxonomy to the Case (Wiki) Folder.
```

```
 75:     Try
 76:         FolderData = FolderAPI.GetFolder(CaseFolderID)
 77:         FolderRequest.FolderId = CaseFolderID
 78:         FolderRequest.BreakInheritButton = True
 79:         FolderRequest.DomainProduction = FolderData.DomainProduction
 80:         FolderRequest.DomainStaging = FolderData.DomainStaging
 81:         FolderRequest.FolderDescription = FolderData.Description
 82:         FolderRequest.MetaInherited = 0
 83:         FolderRequest.FolderName = FolderData.Name
 84:         FolderRequest.IsDomainFolder = FolderData.IsDomainFolder
 85:         FolderRequest.ParentId = FolderData.ParentId
 86:         FolderRequest.PublishActive = FolderData.PublishHtmlActive
 87:         FolderRequest.SiteMapPath = FolderData.SitemapPath
 88:         FolderRequest.SiteMapPathInherit = False
 89:         FolderRequest.StyleSheet = FolderData.StyleSheet
 90:         FolderRequest.TemplateFileName = FolderData.TemplateFileName
 91:         FolderRequest.XmlInherited = FolderData.XmlInherited
 92:         FolderRequest.TaxonomyInherited = False
 93:         FolderRequest.CategoryRequired = True
 94:         FolderRequest.TaxonomyInheritedFrom = CaseFolderID
 95:         FolderRequest.TaxonomyIdList = TaxonomyID.ToString
 96:         FolderAPI.UpdateFolder(FolderRequest)
 97:     Catch ex As Exception
 98:         RemoveCase(CaseFolderID)
 99:         Exit Sub
100:     End Try
101:
102:
103:     'Add default Wiki entry.
104:     Try
105:         DefaultWikiContentID = ContentAPI.AddContent(Me.CaseName & " Wiki", "Default Wiki
Entry", DefaultWikiContentText(), "", "<br/><!-- Wiki Summary -->",
ContentAPI.DefaultContentLanguage, CaseFolderID, "", "", "<metadata><meta id=" & """ &
MetadataDefinitionID & """ & ">Yes</meta></metadata>")
106:
107:     Catch ex As Exception
108:         RemoveCase(CaseFolderID)
109:         Exit Sub
110:     End Try
111:
112:     ' Add default Wiki entry to Taxonomy "general" category.
113:     Try
114:         TaxonomyPath = "\cm." & Me.ClientName & "." & Me.CaseName & "\General"
115:         TaxonomyCategoryID = Taxonomy.GetTaxonomyIdByPath(TaxonomyPath)
116:         TaxonomyContentRequest.ContentId = DefaultWikiContentID
117:         TaxonomyContentRequest.TaxonomyList = Convert.ToString(TaxonomyCategoryID)
118:         ContentAPI.AddTaxonomyItem(TaxonomyContentRequest)
119:     Catch ex As Exception
120:         RemoveCase(CaseFolderID)
121:         Exit Sub
122:     End Try
123:
124:     '-------------------------------------------------------------------------------
125:
126:
127:     '- Documents Folder -
128:     '-------------------------------------------------------------------------------
129:     'Add Case folder
130:     Try
131:         DocumentsFolderID = AddCMCommunityFolder(CaseFolderID, "Documents", "Case
Documents", -1)
132:         'Break Inheritance and set Permissions for Case folder.
133:         CM.SetFolderPermissions(DocumentsFolderID)
134:         'Make Folder is marked 'Private'.
135:         CM.SetFolderPrivate(DocumentsFolderID)
136:         'Set folder template.
```

```
137:         SetFolderTemplate(DocumentsFolderID, "Documents.aspx")
138:     Catch ex As Exception
139:         RemoveCase(CaseFolderID)
140:         Exit Sub
141:     End Try
142:
143:     '-------------------------------------------------------------------------------
144:
145:
146:     '- Milestones Folder -
147:     '-------------------------------------------------------------------------------
148:     Try
149:         'Get Milestones SmartForm ID.
150:         MilestonesSmartFormID = CM.GetSmartFormID()
151:         'Add Case folder.
152:         MilestonesFolderID = AddCMCommunityFolder(CaseFolderID, "Milestones", "Case
Milestones", -1, "StarterApps/CaseManagement/Milestones.aspx", MilestonesSmartFormID)
153:         'Break Inheritance and set Permissions for Case folder.
154:         CM.SetFolderPermissions(MilestonesFolderID)
155:         'Make Folder is marked 'Private'.
156:         CM.SetFolderPrivate(MilestonesFolderID)
157:         'Set Read-Only Permissions on this folder for Membersip users (membership users
cannot add content via smartforms).
158:         MakeFolderReadOnly(MilestonesFolderID)
159:         'Set folder template.
160:         'SetFolderTemplate(MilestonesFolderID, "Milestones.aspx")
161:     Catch ex As Exception
162:         RemoveCase(CaseFolderID)
163:         Exit Sub
164:     End Try
165:
166:
167:
168:     ''Add Milestones SmartForm to Milestones Folder.
169:     'SmartFormCollection.Add(MilestonesSmartFormID, MilestonesSmartFormID)
170:     'ContentRW.UpdateFolderXmlList(MilestonesFolderID, MilestonesSmartFormID,
SmartFormCollection)
171:     '-------------------------------------------------------------------------------
172:
173:     '- Membership User's Blog -
174:     '-------------------------------------------------------------------------------
175:     Try
176:         BlogID = AddBlog(CaseFolderID, "Blog", Me.CaseName & " Blog", "All client members
may view and contribute to this blog")
177:         'Set folder template.
178:         SetFolderTemplate(BlogID, "Blog.aspx")
179:         'Assign "Blog" metadata definition to blog.
180:         MetadataDefinitionID =
CM.GetMetadataDefinitionID(CaseManagement.MetadataDefinitionTypes.MembershipBlog)
181:         AddMetadataDefinitionToFolder(BlogID, MetadataDefinitionID)
182:
183:     Catch ex As Exception
184:         RemoveCase(CaseFolderID)
185:         Exit Sub
186:     End Try
187:     'Add Case Blog.
188:
189:     'Add a default blog entry (Lorem Ipsum).
190:     Try
191:         AddDefaultBlogEntry(BlogID, Me.CaseName & " Blog")
192:     Catch ex As Exception
193:         RemoveCase(CaseFolderID)
194:         Exit Sub
195:     End Try
196:
197:     '-------------------------------------------------------------------------------
```

```
198:
199:     '- CMS User's Blog -
200:     '-------------------------------------------------------------------------------
201:     Try
202:         'Add Private Blog.
203:         BlogID = AddBlog(CaseFolderID, "Private Blog", Me.CaseName & " Private Blog",
"Only CMS users may view and contribute to this blog")
204:         'Remove Membership user group from CMS User's Blog.
205:         UserGroupData = UserGroup.GetUserGroupByName("starterapps.cm." & Me.ClientName)
206:         Permissions.DeleteItemPermission(UserGroupData.GroupId, True, BlogID,
EkEnumeration.CMSObjectTypes.Folder)
207:         'Set folder template.
208:         SetFolderTemplate(BlogID, "Blog.aspx")
209:     Catch ex As Exception
210:         RemoveCase(CaseFolderID)
211:         Exit Sub
212:     End Try
213:
214:     Try
215:         'Assign "PrivateBlog" metadata definition to blog.
216:         MetadataDefinitionID =
CM.GetMetadataDefinitionID(CaseManagement.MetadataDefinitionTypes.CMSBlog)
217:         AddMetadataDefinitionToFolder(BlogID, MetadataDefinitionID)
218:     Catch ex As Exception
219:         RemoveCase(CaseFolderID)
220:         Exit Sub
221:     End Try
222:
223:
224:     'Add a default blog entry (Lorem Ipsum).
225:     Try
226:         AddDefaultBlogEntry(BlogID, Me.CaseName & " Private Blog")
227:     Catch ex As Exception
228:         RemoveCase(CaseFolderID)
229:         Exit Sub
230:     End Try
231:
232:     '-------------------------------------------------------------------------------
233:
234:
235:     '- Issues Folder -
236:     '-------------------------------------------------------------------------------
237:     Try
238:         'Get Issues SmartForm ID.
239:         IssuesSmartFormID = CM.GetIssuesSmartFormID()
240:         'Add Case folder.
241:         IssuesFolderID = AddCMCommunityFolder(CaseFolderID, "Issues", "Case Issues", -1,
"StarterApps/CaseManagement/Issues.aspx", IssuesSmartFormID)
242:         'Break Inheritance and set Permissions for Case folder.
243:         CM.SetFolderPermissions(IssuesFolderID)
244:         'Make Folder is marked 'Private'.
245:         CM.SetFolderPrivate(IssuesFolderID)
246:         'Set Read-Only Permissions on this folder for Membersip users (membership users
cannot add content via smartforms).
247:         MakeFolderReadOnly(IssuesFolderID)
248:         'Set folder template.
249:         SetFolderTemplate(IssuesFolderID, "Issues.aspx")
250:
251:
252:     Catch ex As Exception
253:         RemoveCase(CaseFolderID)
254:         Exit Sub
255:     End Try
256:
257:     'Add Issues SmartForm to Issues Folder.
258:     Try
```

```
259:         SmartFormCollection.Add(IssuesSmartFormID, IssuesSmartFormID)
260:         'ContentRW.UpdateFolderXmlList(IssuesFolderID, IssuesSmartFormID,
SmartFormCollection)
261:     Catch ex As Exception
262:         RemoveCase(CaseFolderID)
263:         Exit Sub
264:     End Try
265:
266:     '-------------------------------------------------------------------------------
267:
268:     '-------------------------------------------------------------------------------
269:
270:     '- Case Discussion -
271:     '-------------------------------------------------------------------------------
272:     Try
273:         DiscussionID = AddDiscussion(CaseFolderID)
274:         'Set folder template.
275:         SetFolderTemplate(DiscussionID, "Discussion.aspx")
276:     Catch ex As Exception
277:         RemoveCase(CaseFolderID)
278:         Exit Sub
279:     End Try
280:
281:     '-------------------------------------------------------------------------------
282: End Sub
```

Cases.AddCMCommunityFolder Method

This method adds a Community folder to the Case Management StarterApp.

C#

```
public int AddCMCommunityFolder(int ParentFolderID, String FolderName, String
FolderDescription, int MetadataDefinitionID);
```

Visual Basic

```
Public Function AddCMCommunityFolder(ByVal ParentFolderID As Integer, ByVal FolderName As
String, ByVal FolderDescription As String, ByVal MetadataDefinitionID As Integer) As Integer
```

Parameters

Parameters	Description
ParentFolderID	The ID of the folder that will become the parent to the folder to be added.
FolderName	The name of the folder to be added.
FolderDescription	The description of the folder to be added. (This can be an empty string.)
MetadataDefinitionID	The ID of a metadata definition to be assigned to the folder.

Returns

Returns the ID of the newly created CMS (see page 1) folder.

Remarks

The metadata definition can be for general purpose but is intended to be used specifically with Wikis. In the Case Management StarterApp, when a user navigates to the Wiki (see page 130), we need to load a content block by default. The first step in figuring out which content block is the default for the selected Wiki (see page 130) is to assign the "cm.DefaultContent" metadata definition to the Wiki (see page 130) folder. Step two is to assign this metadata to a content block in the Wiki (see page 130). Step three is to load the (hopefully) one content block that has been assigned the "cm.DefaultContent" metadata value. Besides that, this method simply creates a Community Folder.

Body Source

```
1: Public Function AddCMCommunityFolder(ByVal ParentFolderID As Integer, ByVal FolderName As
String, ByVal FolderDescription As String, ByVal MetadataDefinitionID As Integer) As Integer
2:     Return AddCMCommunityFolder(ParentFolderID, FolderName, FolderDescription,
MetadataDefinitionID, "", -1)
```

```
3: End Function
```

Cases.AddDefaultBlogEntry Method

This method creates a default blog entry as a placeholder (Lorem Ipsum text).

C#

```
protected AddDefaultBlogEntry(int BlogID, String BlogName);
```

Visual Basic

```
Protected Sub AddDefaultBlogEntry(ByVal BlogID As Integer, ByVal BlogName As String)
```

Parameters

Parameters	Description
BlogID	The ID of the blog where the post will be added.
BlogName	The name of the blog (this is used in the post's title).

Remarks

This method exists to create placeholder text.

Body Source

```
 1: Protected Sub AddDefaultBlogEntry(ByVal BlogID As Integer, ByVal BlogName As String)
 2:     Dim PostContent As New Ektron.Cms.ContentData
 3:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 4:     Dim Categories(0) As String
 5:     Dim NewBlog As New Ektron.Cms.API.Content.Blog
 6:
 7:     'Add Default Blog Entry
 8:     PostContent.ContType = Ektron.Cms.Common.EkConstants.CMSContentType_Content
 9:     PostContent.DateCreated = Now()
10:     PostContent.Html = DefaultBlogContentText()
11:     PostContent.Title = "Default " & BlogName & " Entry"
12:     PostContent.Teaser = LoremIpsumShort()
13:     PostContent.UserId = ContentAPI.UserId
14:     PostContent.LanguageId = ContentAPI.DefaultContentLanguage
15:     PostContent.IsSearchable = True
16:     NewBlog.AddPost(BlogID, PostContent, Categories, False, "", "")
17: End Sub
```

Cases.AddDiscussion Method

This method adds a discussion to the Case Management StarterApp.

C#

```
protected int AddDiscussion(int ParentFolderID);
```

Visual Basic

```
Protected Function AddDiscussion(ByVal ParentFolderID As Integer) As Integer
```

Parameters

Parameters	Description
ParentFolderID	The ID of the folder that will become the parent to the blog to be added.

Returns

The ID of the newly created discussion.

Remarks

Some details regarding how discussions are being created in this method...

1. Name is always "Discussion."

2. Title is always "~Case name~ Discussion."
3. Authentication is set to True.
4. A default category is added ("~Case name~ General Discussion").
5. ModeratePosts is set to False.
6. LockForum is set to False.
7. SortOrder is set to 1. This sets the order of forums added. Since there's only one forum, this param doesn't mean much, but it is required.

Body Source

```
 1: Protected Function AddDiscussion(ByVal ParentFolderID As Integer) As Integer
 2:     Dim Name, Title As String
 3:     Dim Authentication As Boolean
 4:     Dim Categories(0) As String
 5:     Dim BoardID As Integer
 6:     Dim BoardCategories() As Ektron.Cms.DiscussionCategory
 7:     Dim BoardCategory As New Ektron.Cms.DiscussionCategory
 8:     Dim i As Integer = 0
 9:
10:     Name = "Discussion"
11:     Title = Me.CaseName & " Discussion"
12:     Authentication = True
13:     Categories(0) = Me.CaseName & " General Discussion"
14:
15:     Try
16:         Dim Discussion As New Ektron.Cms.API.Content.ThreadedDiscussion
17:         BoardID = Discussion.AddBoard(ParentFolderID, Name, Title, Authentication, "",
Categories)
18:         BoardCategories = Discussion.GetBoardCategories(BoardID)
19:
20:         If Not BoardCategories Is Nothing Then
21:             For Each BoardCategory In BoardCategories
22:                 Discussion.AddForum(BoardID, Me.CaseName & " Forum", Me.CaseName & "
General Discussion", False, False, 1, BoardCategory.CategoryID)
23:             Next
24:         End If
25:     Catch ex As Exception
26:         BoardID = -1
27:     End Try
28:
29:     Return BoardID
30: End Function
```

Cases.AddFolder Method

This method adds a CMS (see page 1) folder to the Case Management StarterApp.

C#

```
protected int AddFolder(int ParentFolderID, String FolderName, String FolderDescription);
```

Visual Basic

```
Protected Function AddFolder(ByVal ParentFolderID As Integer, ByVal FolderName As String,
ByVal FolderDescription As String) As Integer
```

Parameters

Parameters	Description
ParentFolderID	The ID of the folder that will become the parent to the folder to be added.
FolderName	The name of the folder to be added.
FolderDescription	The description of the folder to be added. (This can be an empty string.)

Returns

Returns the ID of the newly created CMS (see page 1) folder.

Remarks

This is intended to be used only when creating the "StarterApps (see page 1)," "CaseManagement (see page 2)," and client folders, but will work for any folder ID (provided user has access to folder). All other content folders in Case Management StarterApp are Community folders.

Body Source

```
 1: Protected Function AddFolder(ByVal ParentFolderID As Integer, ByVal FolderName As String,
ByVal FolderDescription As String) As Integer
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim FolderRequest As New FolderRequest
 4:
 5:     'Create Folder
 6:     FolderRequest.FolderName = FolderName
 7:     FolderRequest.ParentId = ParentFolderID
 8:     FolderRequest.FolderDescription = FolderDescription
 9:     FolderRequest.SiteMapPathInherit = False
10:     FolderRequest.MetaInherited = 0
11:     FolderAPI.AddFolder(FolderRequest)
12:
13:     Return FolderRequest.FolderId
14: End Function
```

Cases.AddMetadataDefinitionToFolder Method

This method assigns a metadata definition (by ID) to a folder (by ID).

C#

```
public AddMetadataDefinitionToFolder(int FolderID, int MetadataDefinitionID);
```

Visual Basic

```
Public Sub AddMetadataDefinitionToFolder(ByVal FolderID As Integer, ByVal MetadataDefinitionID
As Integer)
```

Parameters

Parameters	Description
FolderID	The ID of the folder to assign the metadata definition.
MetadataDefinitionID	The ID of the metadata definition to assign to the folder.

Body Source

```
 1: Public Sub AddMetadataDefinitionToFolder(ByVal FolderID As Integer, ByVal
MetadataDefinitionID As Integer)
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim content As Ektron.Cms.Content.EkContent
 4:     Dim api As New Ektron.Cms.CommonApi
 5:     Dim tmpUserID As Integer = 0
 6:     Dim tmpUserToken As Integer = 0
 7:     tmpUserID = api.RequestInformationRef.CallerId
 8:     tmpUserToken = api.RequestInformationRef.UniqueId
 9:     Try
10:         api.RequestInformationRef.CallerId = EkConstants.InternalAdmin
11:         content = api.EkContentRef
12:         'Assign metadata definition to folder.
13:         content.ProcessCustomFields(FolderID, "false", "Assigned_" & MetadataDefinitionID,
FolderAPI.GetFolder(FolderID).ParentId)
14:     Finally
15:         api.RequestInformationRef.CallerId = tmpUserID
16:         api.RequestInformationRef.UniqueId = tmpUserToken
17:     End Try
```

```
18: End Sub
```

Cases.DefaultBlogContentText Method

Returns default Blog content.

C#

```
private String DefaultBlogContentText();
```

Visual Basic

```
Private Function DefaultBlogContentText() As String
```

Returns

String

Remarks

This method is used to insert content into a content block placeholder when creating a new Blog.

Body Source

```
1: Private Function DefaultBlogContentText() As String
2:     Dim text As StringBuilder = New StringBuilder()
3:     text.Append("<h3>Highlights:</h3><p> </p>")
4:     text.Append("<h3>Status:</h3><p> </p>")
5:     text.Append("<h3>Fires:</h3><p> </p>")
6:
7:     Return text.ToString()
8: End Function
```

Cases.DefaultWikiContentText Method

Returns default Wiki (see page 130) content.

C#

```
private String DefaultWikiContentText();
```

Visual Basic

```
Private Function DefaultWikiContentText() As String
```

Returns

String

Remarks

This method is used to insert content into a content block placeholder when creating a new Wiki (see page 130).

Body Source

```
1: Private Function DefaultWikiContentText() As String
2:     Dim text As StringBuilder = New StringBuilder()
3:     text.Append("<h3>Stake Holder:</h3><p> </p>")
4:     text.Append("<h3>Case Goals:</h3><p> </p>")
5:     text.Append("<h3>Success Criteria:</h3><p> </p>")
6:
7:     Return text.ToString()
8: End Function
```

Cases.GetAllCases Method

This method retrieves all Case folder names and IDs that the user has permissions to view.

C#

```
public NameValueCollection GetAllCases();
```

Visual Basic

```
Public Function GetAllCases() As NameValueCollection
```

Returns

A NameValueCollection made up of the folder's name and the folder's ID.

Remarks

This method respects permissions (using the GetChildFolders() method) and only returns the folders the user is allowed to view.

Body Source

```
 1: Public Function GetAllCases() As NameValueCollection
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim CM As New Ektron.Cms.StarterApps.CaseManagement.CaseManagement
 4:     Dim CMFolderID As Integer
 5:     Dim Cases As New NameValueCollection
 6:     Dim FolderData() As FolderData
 7:     Dim Folder As New FolderData
 8:
 9:     'Get the Case Management folder's ID.
10:     CMFolderID = CM.Initialize()
11:     'Create nameValueCollection of folder names and folder IDs.
12:     FolderData = FolderAPI.GetChildFolders(CMFolderID, False)
13:     If Not FolderData Is Nothing Then
14:         For Each Folder In FolderData
15:             Cases.Add(Folder.Name, Folder.Id)
16:         Next
17:     End If
18:     'Return the NameValueCollection.
19:     Return Cases
20: End Function
```

Cases.GetCaseTaxonomyID Method

Gets the ID of the taxonomy associated with the Case folder.

C#

```
public int GetCaseTaxonomyID(int CaseFolderID);
```

Visual Basic

```
Public Function GetCaseTaxonomyID(ByVal CaseFolderID As Integer) As Integer
```

Parameters

Parameters	Description
CaseFolderID	FolderID of the Case folder.

Returns

Integer

Remarks

This method returns the taxonomy ID associated with the Case ("Wiki (see page 130)").

Body Source

```
 1: Public Function GetCaseTaxonomyID(ByVal CaseFolderID As Integer) As Integer
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim FolderData As New Ektron.Cms.FolderData
 4:     Dim Taxonomy As New Ektron.Cms.API.Content.Taxonomy
 5:     Dim TaxonomyBaseData() As Ektron.Cms.TaxonomyBaseData
 6:     Dim TaxonomyBaseDatum As New Ektron.Cms.TaxonomyBaseData
 7:     Dim CaseName, ClientName As String
 8:     Dim TaxonomyID As Integer = -1
```

```
 9:
10:     '    'Get the Case's name.
11:     FolderData = FolderAPI.GetFolder(CaseFolderID, True)
12:     CaseName = System.Web.HttpUtility.HtmlDecode(FolderData.Name)
13:     TaxonomyBaseData = FolderData.FolderTaxonomy
14:
15:     'Get the client's name.
16:     FolderData = FolderAPI.GetFolder(FolderData.ParentId)
17:     ClientName = System.Web.HttpUtility.HtmlDecode(FolderData.Name)
18:
19:     'Get the Case's taxonomy.
20:     For Each TaxonomyBaseDatum In TaxonomyBaseData
21:         If TaxonomyBaseDatum.TaxonomyName = "cm." & ClientName & "." & CaseName Then
22:             TaxonomyID = TaxonomyBaseDatum.TaxonomyId
23:             Exit For
24:         End If
25:     Next
26:
27:     Return TaxonomyID
28: End Function
```

Cases.GetDefaultTaxonomy Method

This method loads the default taxonomy ("DefaultCaseTaxonomy.xml") into a string.

C#

```
private String GetDefaultTaxonomy();
```

Visual Basic

```
Private Function GetDefaultTaxonomy() As String
```

Returns

An Taxonomy XML String.

Remarks

This method is used during Case creation. The LoadTaxonomy() method takes an xml string as a param. We grab the default taxonomy, load it as a System.Xml.XmlDocument object, and return InnerXML.

Body Source

```
1: Private Function GetDefaultTaxonomy() As String
2:     Dim DefaultTaxonomy As New System.Xml.XmlDocument
3:
4:
DefaultTaxonomy.Load(System.Web.HttpContext.Current.Server.MapPath("../xml/DefaultCaseTaxonomy.
xml"))
5:     GetDefaultTaxonomy = DefaultTaxonomy.InnerXml
6:
7:     Return GetDefaultTaxonomy
8: End Function
```

Cases.LoremIpsumLong Method

This method generates placeholder text.

C#

```
private String LoremIpsumLong();
```

Visual Basic

```
Private Function LoremIpsumLong() As String
```

Returns

3 html paragraphs of Lorem Ipsum text.

Remarks

This is here to save the hassle of cutting and pasting placeholder text.

Body Source

```
1: Private Function LoremIpsumLong() As String
2:     LoremIpsumLong = "<p>" & "Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Donec pharetra, nisl tristique semper rutrum, risus magna congue metus, faucibus venenatis
orci metus vel lectus. Duis at ante. Donec bibendum. In lectus orci, rhoncus hendrerit, cursus
dictum, auctor pharetra, tortor. Maecenas metus mi, mollis non, iaculis eget, ornare in,
risus. Cras ut odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere
cubilia Curae; Vestibulum eu lorem. Suspendisse quis mauris. Quisque nisl. Maecenas felis.
Fusce dui odio, tristique sit amet, tincidunt et, volutpat a, diam. Phasellus non libero.
Phasellus et leo a libero cursus congue. Pellentesque feugiat est nec nulla." & "</p>"
3:     LoremIpsumLong = LoremIpsumLong & "<p>" & "Proin et enim interdum nisl elementum
porttitor. Etiam quam turpis, hendrerit quis, dictum id, ultricies quis, tortor. Vestibulum
ante metus, aliquet quis, posuere at, pharetra at, lorem. Etiam nec felis. Integer fringilla
bibendum arcu. Pellentesque iaculis libero vitae libero. Aenean euismod mollis lorem. Etiam
sit amet arcu. Nunc nisi dolor, luctus vel, rhoncus commodo, tempor ut, turpis. Praesent
auctor vehicula erat. Sed porttitor accumsan massa. Phasellus sed lectus." & "</p>"
4:     LoremIpsumLong = LoremIpsumLong & "<p>" & "Nunc porta ornare pede. Donec tincidunt
elementum mauris. Aliquam interdum elit id neque. Donec quis eros ac tellus nonummy laoreet.
Nam ultrices, risus sit amet molestie dignissim, dolor turpis tempus est, in semper dui risus
vel nulla. Morbi iaculis metus at ipsum. Praesent lectus pede, malesuada eget, bibendum quis,
bibendum luctus, orci. Pellentesque rhoncus lacus a mi. Nam vulputate lorem eget neque.
Aliquam aliquet purus in erat. Vivamus dignissim fringilla enim. Curabitur id magna. Maecenas
aliquet posuere ante. Proin arcu tortor, cursus in, facilisis quis, sagittis sit amet, est.
Nunc et erat at ipsum semper ultricies. Sed mi. Vestibulum sed nisi. Donec massa nisl, posuere
sed, pharetra et, egestas vitae, magna." & "</p>"
5:     Return LoremIpsumLong
6: End Function
```

Cases.LoremIpsumShort Method

This method generates placeholder text.

C#

```
private String LoremIpsumShort();
```

Visual Basic

```
Private Function LoremIpsumShort() As String
```

Returns

1 html paragraphs of Lorem Ipsum text.

Remarks

This is here to save the hassle of cutting and pasting placeholder text.

Body Source

```
1: Private Function LoremIpsumShort() As String
2:     LoremIpsumShort = "<p>" & "Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Donec pharetra, nisl tristique semper rutrum, risus magna congue metus, faucibus venenatis
orci metus vel lectus. Duis at ante. Donec bibendum. In lectus orci, rhoncus hendrerit, cursus
dictum, auctor pharetra, tortor. Maecenas metus mi, mollis non, iaculis eget, ornare in,
risus. Cras ut odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere
cubilia Curae; Vestibulum eu lorem. Suspendisse quis mauris. Quisque nisl. Maecenas felis.
Fusce dui odio, tristique sit amet, tincidunt et, volutpat a, diam. Phasellus non libero.
Phasellus et leo a libero cursus congue. Pellentesque feugiat est nec nulla." & "</p>"
3:     Return LoremIpsumShort
4: End Function
```

Cases.MakeFolderNameUnique Method

This method ensures a Case name is unique.

C#

```
private String MakeFolderNameUnique(int ParentFolderID, String NewFolderName);
```

Visual Basic

```
Private Function MakeFolderNameUnique(ByVal ParentFolderID As Integer, ByVal NewFolderName As
String) As String
```

Parameters

Parameters	Description
ParentFolderID	Folder ID of the parent folder (the "client" folder).
NewFolderName	The desired name of the child folder (the "Case" folder).

Returns

The unique name.

Remarks

This method checks to see if there is a sibling folder that has already been given the name passed in the parameter "NewFolderName." If a folder already exists with this name, the function appends to the name a *count* like Windows does.

For example, If NewFolderName = "Ektron" and a folder named "Ektron" already exists, this method returns "Ektron (1)."

Body Source

```
 1: Private Function MakeFolderNameUnique(ByVal ParentFolderID As Integer, ByVal NewFolderName
As String) As String
 2:     Dim UniqueName As String = NewFolderName
 3:     Dim IsUnique As Boolean = True
 4:     Dim i As Integer = 1
 5:     Dim currentComponentName As String
 6:     Dim FolderAPI As New Ektron.Cms.API.Folder
 7:
 8:     Dim SiblingFolderData() As FolderData
 9:     Dim SiblingFolder As FolderData
10:
11:     SiblingFolderData = FolderAPI.GetChildFolders(ParentFolderID)
12:     If Not SiblingFolderData Is Nothing Then
13:         For Each SiblingFolder In SiblingFolderData
14:             If SiblingFolder.Name = NewFolderName Then
15:                 IsUnique = False
16:             End If
17:         Next
18:     End If
19:
20:     If IsUnique = False Then
21:         For Each SiblingFolder In SiblingFolderData
22:             currentComponentName = NewFolderName & " (" & i & ")"
23:             If currentComponentName = SiblingFolder.Name Then
24:                 i = i + 1
25:             End If
26:         Next
27:         UniqueName = NewFolderName & " (" & i & ")"
28:     End If
29:
30:     Return UniqueName
31: End Function
```

Cases.MakeFolderReadOnly Method

This method is for use with the Milestones folder in the Case Management Starter App.

C#

```
private MakeFolderReadOnly(int FolderID);
```

Visual Basic

```
Private Sub MakeFolderReadOnly(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	This should be the ID of the "Milestones" folder.

Remarks

The "Milestones" folder is read-only for Membership users, because to add content, you must use a Smart Form. Membership users cannot add content via Smart Forms. Thus, this folder is read-only for Membership users.

Body Source

```
 1: Private Sub MakeFolderReadOnly(ByVal FolderID As Integer)
 2:     Dim User As New Ektron.Cms.API.User.User
 3:     Dim UserGroup As New Ektron.Cms.UserGroupData
 4:     Dim Users As New Ektron.Cms.StarterApps.CaseManagement.Users
 5:     Dim MembershipGroupID As Integer
 6:     Dim UserPermissionData As New Ektron.Cms.UserPermissionData
 7:     Dim Permissions As New Ektron.Cms.API.Permissions
 8:
 9:     'Get the Membership user group's ID.
10:     UserGroup = User.GetUserGroupByName("starterapps.cm." & Me.ClientName)
11:     MembershipGroupID = UserGroup.GroupId
12:
13:     'Set Read-Only permissions.
14:     UserPermissionData.FolderId = FolderID
15:     UserPermissionData.UserId = -1
16:     UserPermissionData.GroupId = MembershipGroupID
17:     UserPermissionData.IsReadOnly = True
18:     UserPermissionData.CanAdd = False
19:     UserPermissionData.CanEdit = False
20:     UserPermissionData.IsReadOnlyLib = True
21:     UserPermissionData.CanAddToImageLib = False
22:     UserPermissionData.CanAddToFileLib = False
23:     'Update the folder.
24:     Try
25:         Permissions.UpdateItemPermission(UserPermissionData)
26:     Catch ex As Exception
27:
28:     End Try
29:
30: End Sub
```

Cases.RemoveCase Method

This method removes a Case from the Case Management StarterAPP.

C#

```
public RemoveCase(int _CaseFolderID);
```

Visual Basic

```
Public Sub RemoveCase(ByVal _CaseFolderID As Integer)
```

Parameters

Parameters	Description
CaseFolderID	The ID of the Case (folder ID) that you wish to remove.

Remarks

This method removes a Case and all Case contents including Wiki (see page 130) content (and the associated Wiki (see page 130) taxonomy), discussion content, documents and blogs.

Body Source

```
 1: Public Sub RemoveCase(ByVal _CaseFolderID As Integer)
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim FolderData As New Ektron.Cms.FolderData
 4:     Dim CaseName, ClientName As String
 5:     Dim Taxonomy As New Ektron.Cms.API.Content.Taxonomy
 6:     Dim TaxonomyBaseData() As Ektron.Cms.TaxonomyBaseData
 7:     Dim TaxonomyBaseDatum As New Ektron.Cms.TaxonomyBaseData
 8:     Dim TaxonomyRequest As New Ektron.Cms.TaxonomyRequest
 9:
10:     'Get the Case's name.
11:     FolderData = FolderAPI.GetFolder(_CaseFolderID, True)
12:     CaseName = FolderData.Name
13:     TaxonomyBaseData = FolderData.FolderTaxonomy
14:
15:     'Get the client's name.
16:     FolderData = FolderAPI.GetFolder(FolderData.ParentId)
17:     ClientName = FolderData.Name
18:
19:     'Delete the Case's taxonomy.
20:     For Each TaxonomyBaseDatum In TaxonomyBaseData
21:         If TaxonomyBaseDatum.TaxonomyName = "cm." &
Ektron.Cms.Common.EkFunctions.HtmlDecode(ClientName) & "." &
Ektron.Cms.Common.EkFunctions.HtmlDecode(CaseName) Then
22:             TaxonomyRequest.TaxonomyId = TaxonomyBaseDatum.TaxonomyId
23:             TaxonomyRequest.TaxonomyLanguage = FolderAPI.DefaultContentLanguage
24:             Taxonomy.DeleteTaxonomy(TaxonomyRequest)
25:             Exit For
26:         End If
27:     Next
28:
29:     'Delete the Case's folder.
30:     Try
31:         FolderAPI.DeleteFolderById(_CaseFolderID)
32:     Catch ex As Exception
33:         Throw New Exception("Cannot Delete Case Folder: " & ex.Message)
34:     End Try
35: End Sub
```

Cases.SetFolderTemplate Method

This method sets the default .aspx template for a folder.

C#

```
public SetFolderTemplate(int FolderID, String TemplateName);
```

Visual Basic

```
Public Sub SetFolderTemplate(ByVal FolderID As Integer, ByVal TemplateName As String)
```

Parameters

Parameters	Description
FolderID	The ID of the folder to which the template is assigned.
TemplateName	The name of the template to assign to the folder.

Remarks

In addition to assigning the template specified in the parameter, this method also sets the folder's associated stylesheet to "nothing.css." This is done to ensure all css styles are applied via the css files (and styles) specified in the html head section.

That is, so unwanted styles don't sneak in uninvited.

Body Source

```
 1: Public Sub SetFolderTemplate(ByVal FolderID As Integer, ByVal TemplateName As String)
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim FolderData As New Ektron.Cms.FolderData
 4:     Dim FolderRequest As New Ektron.Cms.FolderRequest
 5:
 6:     FolderData = FolderAPI.GetFolder(FolderID, True)
 7:     FolderRequest.FolderId = FolderID
 8:     FolderRequest.BreakInheritButton = True
 9:     FolderRequest.DomainProduction = FolderData.DomainProduction
10:     FolderRequest.DomainStaging = FolderData.DomainStaging
11:     FolderRequest.FolderDescription = FolderData.Description
12:     FolderRequest.MetaInherited = 0
13:     FolderRequest.FolderName = FolderData.Name
14:     FolderRequest.IsDomainFolder = FolderData.IsDomainFolder
15:     FolderRequest.ParentId = FolderData.ParentId
16:     FolderRequest.PublishActive = FolderData.PublishHtmlActive
17:     FolderRequest.SiteMapPath = FolderData.SitemapPath
18:     FolderRequest.SiteMapPathInherit = False
19:     FolderRequest.StyleSheet = "StarterApps/CaseManagement/css/ewebeditprostyles.css"
20:     FolderRequest.TemplateFileName = "StarterApps/CaseManagement/" & TemplateName
21:     FolderRequest.XmlInherited = FolderData.XmlInherited
22:     FolderAPI.UpdateFolder(FolderRequest)
23: End Sub
```

Cases Properties

The properties of the Cases class are listed here.

Public Properties

	Name	Description
	CaseName (see page 45)	The CaseName property is the name associated with the Case Folder.
	ClientID (see page 45)	The ID of the folder associated with the selected client.
	ClientName (see page 46)	The name of the selected client as an unescaped string.
	CMFolderID (see page 46)	The Integer ID of the Case management folder.

Legend

	Property

Cases.CaseName Property

The CaseName property is the name associated with the Case Folder.

C#

```
public String CaseName;
```

Visual Basic

```
Public Property CaseName() As String
```

Returns

Returns unescaped Case name.

Cases.ClientID Property

The ID of the folder associated with the selected client.

C#

```
public int ClientID;
```

Visual Basic

```
Public Property ClientID() As Integer
```

Returns

The FolderID of the selected client.

Cases.ClientName Property

The name of the selected client as an unescaped string.

C#

```
public String ClientName;
```

Visual Basic

```
Public Property ClientName() As String
```

Returns

The name of the client.

Cases.CMFolderID Property

The Integer ID of the Case management folder.

C#

```
public int CMFolderID;
```

Visual Basic

```
Public Property CMFolderID() As Integer
```

Returns

The ID of the CM Folder.

Clients Class

Contains methods for managing clients in the Case Manager Starter Application.

Class Hierarchy

Ektron.Cms.StarterApps.CaseManagement.Clients

C#

```
public class Clients;
```

Visual Basic

```
Public Class Clients
```

File

CaseManagement.vb

Clients Methods

The methods of the Clients class are listed here.

Public Methods

	Name	Description
	AddClient (see page 47)	This method adds a new client to the Case Management. StarterApp.

	GetAllClients (see page 48)	This method gathers all the client names and client folder ID's that the current user has permissions to view.
	GetClientXMLForInformationArchitecture (see page 49)	This method gathers the Information Architecture for a particular client.
	RemoveClient (see page 50)	This method removes a client from the Case Management StarterApp.

Legend

	Method

Clients.AddClient Method

This method adds a new client to the Case Management. StarterApp.

C#

```
public int AddClient(String ClientName);
```

Visual Basic

```
Public Function AddClient(ByVal ClientName As String) As Integer
```

Parameters

Parameters	Description
ClientName	The name of the client to add.

Returns

The ID of the newly created Client folder.

Remarks

This method does several things besides creating a CMS (see page 1) folder for a client.

1. If the client name already exists it doesn't do anything.
2. It breaks both content and metadata inheritance on the folder - this is so we can set the exact permissions and metadata models we wish.
3. It sets the folder to PRIVATE.
4. It creates a new Membership user group ("cm.~client name~").
5. It adds the "starterapps.cm" CMS (see page 1) user group to the folder.

Body Source

```
 1: Public Function AddClient(ByVal ClientName As String) As Integer
 2:     Dim cm As New Ektron.Cms.StarterApps.CaseManagement.CaseManagement
 3:     Dim CMFolderID As Integer
 4:     Dim ClientList As New NameValueCollection
 5:     Dim ClientExists As Boolean = False
 6:     Dim ClientFolderID As Integer
 7:     Dim ClientFolder As New FolderRequest
 8:     Dim Cases As New Cases
 9:     Dim Users As New Ektron.Cms.StarterApps.CaseManagement.Users
10:     Dim MembershipUserGroupID As Integer
11:     Dim FolderAPI As New Ektron.Cms.API.Folder
12:
13:     'Unescape Client Name.
14:     ClientName = System.Web.HttpUtility.HtmlDecode(ClientName)
15:
16:     'Get the Case Management folder ID.
17:     CMFolderID = cm.Initialize()
18:
19:     'Get all client folders inside the Case Management Folder.
20:     ClientList = GetAllClients()
21:
22:     'Loop to see if the client folder already exists.
```

```
    If Not ClientList.Count = 0 Then
        Dim i As Integer
        For i = 0 To ClientList.Count - 1
            If System.Web.HttpUtility.HtmlDecode(ClientList.GetKey(i)) = ClientName Then
                ClientExists = True
                ClientFolderID = ClientList.GetValues(i)(0)
                Exit For
            End If
        Next
    End If

    'If client does not exist, create the client folder.
    If ClientExists = False Then

        'Create Client Folder.
        ClientFolder.FolderName = ClientName
        ClientFolder.ParentId = CMFolderID
        ClientFolder.FolderDescription = ClientName & " Case Management Folder"
        ClientFolder.SiteMapPathInherit = False
        ClientFolder.MetaInherited = 0
        FolderAPI.AddCommunityFolder(ClientFolder)

        'Get the client ID.
        ClientFolderID = ClientFolder.FolderId
        'Break inheritance.
        cm.SetFolderPermissions(ClientFolderID)
        'Make the Folder Private.
        cm.SetFolderPrivate(ClientFolderID)
        'Add MembershipUserGroup.
        MembershipUserGroupID = Users.AddMembershipUserGroup("starterapps.cm." &
ClientName)
        'Add MembershipUserGroup To Client Folder.
        If MembershipUserGroupID <> -1 Then
            Users.AddMembershipUserGroupToFolder(ClientFolderID, MembershipUserGroupID)
        End If
    Else
        ClientFolderID = -2
    End If

    'Return the ID of the client folder.
    Return ClientFolderID
End Function
```

Clients.GetAllClients Method

This method gathers all the client names and client folder ID's that the current user has permissions to view.

C#

```
public NameValueCollection GetAllClients();
```

Visual Basic

```
Public Function GetAllClients() As NameValueCollection
```

Returns

NameValueCollection - client name, client folder ID.

Remarks

This method uses GetChildFolders and therefore respects permissions.

Body Source

```
Public Function GetAllClients() As NameValueCollection
    Dim cm As New Ektron.Cms.StarterApps.CaseManagement.CaseManagement
    Dim CMFolderID As Integer
    Dim cmFolderData() As FolderData
```

```
 5:     Dim cmFolderItem As New FolderData
 6:     Dim Clients As New NameValueCollection
 7:     Dim FolderAPI As New Ektron.Cms.API.Folder
 8:
 9:     CMFolderID = cm.Initialize()
10:
11:     'Create nameValueCollection of folder names and folder IDs.
12:     cmFolderData = FolderAPI.GetChildFolders(CMFolderID, False)
13:     If Not cmFolderData Is Nothing Then
14:         For Each cmFolderItem In cmFolderData
15:             Clients.Add(cmFolderItem.Name, cmFolderItem.Id)
16:         Next
17:     End If
18:
19:     Return Clients
20: End Function
```

Clients.GetClientXMLForInformationArchitecture Method

This method gathers the Information Architecture for a particular client.

C#

```
public System.Xml.XmlDocument GetClientXMLForInformationArchitecture(int ClientID);
```

Visual Basic

```
Public Function GetClientXMLForInformationArchitecture(ByVal ClientID As Integer) As
System.Xml.XmlDocument
```

Parameters

Parameters	Description
`ClientID`	The ID of the client (folder ID) for which the IA is to be generated.

Returns

An XML document containing the Information Architecture for a particular client.

Remarks

This method is a more efficient version of Case.GetInformationArchitectureXML(), which returns the entire IA tree for a particular user. This method restricts the tree to a particular branch (one client), so it's a bit more efficient. Case.GetInformationArchitect() also has a boolean switch to include all the members of the client's Membership user group. This method does not have such a switch.

Body Source

```
 1: Public Function GetClientXMLForInformationArchitecture(ByVal ClientID As Integer) As
System.Xml.XmlDocument
 2:
 3:     Dim cm As New Ektron.Cms.StarterApps.CaseManagement.CaseManagement
 4:     Dim DocumentNode, ClientNode As System.Xml.XmlNode
 5:     Dim XMLUtils As New Ektron.Cms.StarterApps.CaseManagement.XMLUtilities
 6:     Dim FolderAPI As New Ektron.Cms.API.Folder
 7:     Dim ClientFolderData As New Ektron.Cms.FolderData
 8:     Dim FolderData As New Ektron.Cms.FolderData
 9:
10:     GetClientXMLForInformationArchitecture = New System.Xml.XmlDocument
11:
12:     FolderData = FolderAPI.GetFolder(ClientID)
13:
14:     'Initalize the IA XML document.
15:     DocumentNode = GetClientXMLForInformationArchitecture.CreateElement("ektron")
16:     GetClientXMLForInformationArchitecture.AppendChild(DocumentNode)
17:
18:     ClientNode = GetClientXMLForInformationArchitecture.CreateElement("li")
19:     XMLUtils.ApplyXMLAttribute(GetClientXMLForInformationArchitecture, ClientNode, "id",
```

```
ClientID)
20:     XMLUtils.ApplyXMLAttribute(GetClientXMLForInformationArchitecture, ClientNode,
"label", FolderData.Name)
21:     XMLUtils.ApplyXMLAttribute(GetClientXMLForInformationArchitecture, ClientNode,
"class", "Client")
22:     GetClientXMLForInformationArchitecture.DocumentElement.AppendChild(ClientNode)
23:
24:     Return GetClientXMLForInformationArchitecture
25: End Function
```

Clients.RemoveClient Method

This method removes a client from the Case Management StarterApp.

C#

```
public RemoveClient(int ClientFolderID);
```

Visual Basic

```
Public Sub RemoveClient(ByVal ClientFolderID As Integer)
```

Parameters

Parameters	Description
ClientFolderID	The ID of the client (folder ID) to be removed.

Remarks

1. This method calls Cases.RemoveCase (see page 43)() to individually remove each Case to which this client is a parent.Case.
2. This method removes the client Membership group "cm.~client name~."
3. This method removes the client folder.

Body Source

```
 1: Public Sub RemoveClient(ByVal ClientFolderID As Integer)
 2:     Dim UserGroup As New Ektron.Cms.StarterApps.CaseManagement.Users
 3:     Dim Cases As New Ektron.Cms.StarterApps.CaseManagement.Cases
 4:     Dim CaseFolders() As Ektron.Cms.FolderData
 5:     Dim CaseFolderID As Integer
 6:     Dim FolderAPI As New Ektron.Cms.API.Folder
 7:
 8:     'Delete all of the client's Cases
 9:     CaseFolders = FolderAPI.GetChildFolders(ClientFolderID)
10:     If CaseFolders IsNot Nothing Then
11:         For Each CaseFolder As FolderData In CaseFolders
12:             CaseFolderID = CaseFolder.Id
13:             Try
14:                 Cases.RemoveCase(CaseFolderID)
15:             Catch ex As Exception
16:
17:             End Try
18:
19:         Next
20:     End If
21:
22:
23:     'Delete client Membership user group.
24:     Try
25:         UserGroup.RemoveUserGroup("starterapps.cm." &
FolderAPI.GetFolder(ClientFolderID).Name)
26:     Catch ex As Exception
27:
28:     End Try
29:
30:
```

```
31:     'Delete client folder.
32:     Try
33:         FolderAPI.DeleteFolderById(ClientFolderID)
34:     Catch ex As Exception
35:
36:     End Try
37:
38:
39:
40: End Sub
```

Users Class

Contains methods for managing users in the Case Manager Starter Application.

Class Hierarchy

Ektron.Cms.StarterApps.CaseManagement.Users

C#

```
public class Users;
```

Visual Basic

```
Public Class Users
```

File

CaseManagement.vb

Users Methods

The methods of the Users class are listed here.

Public Methods

	Name	Description
	AddCMSUserGroup (see page 51)	This method adds a CMS (see page 1) user group to the Case Management StarterApp.
	AddMembershipUserGroup (see page 52)	This method adds a Membership user group to the Case Management StarterApp.
	AddMembershipUserGroupToFolder (see page 53)	This method adds a Membership user group to a folder and sets the permissions appropriate to the Case Management Starer App for the group on the folder.
	AddMemberToClient (see page 54)	This method adds a user (by user ID) to a user group (by user group ID).
	AddUserGroupToFolder (see page 55)	This method adds a user group to a folder and sets Permissions for the group on the folder.
	AddUserToGroup (see page 56)	This method adds a user (by user ID) to a user group (by group name).
	RemoveMemberFromClient (see page 57)	This method removes a Membership user from a client user group.
	RemoveUserGroup (see page 57)	This method removes a user group (by user group name).

Legend

	Method

Users.AddCMSUserGroup Method

This method adds a CMS (see page 1) user group to the Case Management StarterApp.

C#

```
public int AddCMSUserGroup(String CMSGroupName);
```

Visual Basic

```
Public Function AddCMSUserGroup(ByVal CMSGroupName As String) As Integer
```

Parameters

Parameters	Description
CMSGroupName	The name of the CMS (see page 1) group to add.

Returns

The ID of the newly created CMS (see page 1) user group.

Remarks

This method is meant to be executed only when the StarterApp initializes itself. There is only 1 CMS (see page 1) user group in the StarterApp by design. All members of the CMS (see page 1) user group have permissions to do everything to all folders within the StarterApp.

Body Source

```
 1: Public Function AddCMSUserGroup(ByVal CMSGroupName As String) As Integer
 2:     Dim User As New Ektron.Cms.API.User.User
 3:     Dim UserGroups() As UserGroupData
 4:     Dim UserGroup As New UserGroupData
 5:     Dim UserGroupExists As Boolean = False
 6:     Dim CMSGroupID As Integer
 7:
 8:     'See if the CMS user group name already exists.
 9:     UserGroups = User.GetAllUserGroups(EkEnumeration.UserTypes.AuthorType)
10:     For Each UserGroup In UserGroups
11:         If UserGroup.GroupName = CMSGroupName Then
12:             CMSGroupID = UserGroup.GroupId
13:             UserGroupExists = True
14:             Exit For
15:         End If
16:     Next
17:
18:     'If the CMS user group doesn't exist, create it.
19:     If UserGroupExists = False Then
20:         User.AddUserGroup(CMSGroupName)
21:         UserGroup = User.GetUserGroupByName(CMSGroupName)
22:         CMSGroupID = UserGroup.GroupId
23:     End If
24:
25:     'return the CMS user group ID.
26:     Return CMSGroupID
27: End Function
```

Users.AddMembershipUserGroup Method

This method adds a Membership user group to the Case Management StarterApp.

C#

```
public int AddMembershipUserGroup(String MemershipGroupName);
```

Visual Basic

```
Public Function AddMembershipUserGroup(ByVal MemershipGroupName As String) As Integer
```

Parameters

Parameters	Description
MemershipGroupName	The name of the Membership user group to add.

Returns

The ID of the newly created Membership user group.

Remarks

This method is executed once per client. That is, each client gets its own Membership user group. The client's Membership user

group is deleted when the client is removed with the RemoveClient() method.

Body Source

```
 1: Public Function AddMembershipUserGroup(ByVal MemershipGroupName As String) As Integer
 2:     Dim MembershipUserGroupCollection As New Collection
 3:     Dim User As New Ektron.Cms.API.User.User
 4:     Dim UserGroups() As UserGroupData
 5:     Dim UserGroup As New UserGroupData
 6:     Dim UserGroupExists As Boolean = False
 7:     Dim MembershipGroupID As Integer
 8:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 9:     Dim ErrorMessage As String = ""
10:
11:     'See if the CMS user group name already exists.
12:     UserGroups = User.GetAllUserGroups(EkEnumeration.UserTypes.AuthorType)
13:     For Each UserGroup In UserGroups
14:         If UserGroup.GroupName = MemershipGroupName Then
15:             MembershipGroupID = UserGroup.GroupId
16:             UserGroupExists = True
17:             Exit For
18:         End If
19:     Next
20:
21:     'If user group doesn't exist, create it.
22:     If UserGroupExists = False Then
23:         MembershipUserGroupCollection.Add(MemershipGroupName, "UserGroupName")
24:         MembershipUserGroupCollection.Add("StarterApp membership usergroup", "Description")
25:         Try
26:             Dim tmpID As Integer = ContentAPI.RequestInformationRef.CallerId
27:             ContentAPI.RequestInformationRef.CallerId = EkConstants.InternalAdmin
28:             ContentAPI.EkUserRef.AddMemberShipGroupV4(MembershipUserGroupCollection, "",
""")
29:             ContentAPI.RequestInformationRef.CallerId = tmpID
30:
31:             UserGroup = User.GetUserGroupByName(MemershipGroupName)
32:             MembershipGroupID = UserGroup.GroupId
33:         Catch ex As Exception
34:             MembershipGroupID = -1
35:             ErrorMessage = ErrorMessage & "Membership Group Could Not Be Created.  "
36:             ErrorMessage = ErrorMessage & "You may manually work around this error by
creating a Membership usergroup in the Workarea.  "
37:             ErrorMessage = ErrorMessage & "The Membership usergroup name must be '" &
MemershipGroupName & "'."
38:             Throw New Exception(ErrorMessage & ex.Message)
39:         End Try
40:
41:     End If
42:
43:     'return the user group ID.
44:     Return MembershipGroupID
45: End Function
```

Users.AddMembershipUserGroupToFolder Method

This method adds a Membership user group to a folder and sets the permissions appropriate to the Case Management Starer App for the group on the folder.

C#

```
public AddMembershipUserGroupToFolder(int FolderID, int GroupID);
```

Visual Basic

```
Public Sub AddMembershipUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer)
```

Parameters

Parameters	Description
FolderID	The folder to which you wish to assign the user group.
GroupID	The group ID of the Membership user group you wish to assign to the folder.

Remarks

This method includes the permission settings appropriate to Membership users (see page 51) in the Case Management Starter App.

Body Source

```
 1: Public Sub AddMembershipUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As
Integer)
 2:     Dim UserPermissionData As New Ektron.Cms.UserPermissionData
 3:     Dim Permissions As New Ektron.Cms.API.Permissions
 4:
 5:     'Add permissions for GroupID to FolderID.
 6:     UserPermissionData.GroupId = GroupID
 7:     UserPermissionData.FolderId = FolderID
 8:     'File/folder permissions.
 9:     UserPermissionData.CanAdd = True
10:     UserPermissionData.CanEdit = True
11:     UserPermissionData.CanDelete = False
12:     'Library permissions.
13:     UserPermissionData.CanAddToFileLib = True
14:     UserPermissionData.CanAddToImageLib = True
15:     UserPermissionData.CanAddToHyperlinkLib = False
16:     UserPermissionData.CanOverwriteLib = False
17:     'Membership users can traverse folders by default (hardcoded to True).
18:     'We are "adding" here to show explicitly that this permission setting is required.
19:     UserPermissionData.CanTraverseFolders = True
20:     Try
21:         Permissions.AddItemPermission(UserPermissionData)
22:     Catch ex As Exception
23:
24:     End Try
25:
26: End Sub
```

Users.AddMemberToClient Method

This method adds a user (by user ID) to a user group (by user group ID).

C#

```
public AddMemberToClient();
```

Visual Basic

```
Public Sub AddMemberToClient()
```

Remarks

This method requires population of two Users (see page 51) properties...

1. MemberID (see page 58)
2. ClientID (see page 58)

This method is more or less the same thing as AddUserToGroup (see page 56)() only this method takes a user group ID whereas AddUserToGroup (see page 56)() takes a user group name.

Body Source

```
 1: Public Sub AddMemberToClient()
 2:     Dim Member As New Ektron.Cms.API.User.User
 3:     Dim GroupInfo As New UserGroupData
```

```
 4:     Dim Members As New Ektron.Cms.API.User.User
 5:     Dim FolderAPI As New Ektron.Cms.API.Folder
 6:
 7:     GroupInfo = Member.GetUserGroupByName("starterapps.cm." &
FolderAPI.GetFolder(Me.ClientID).Name)
 8:     Try
 9:         Members.AddUserToGroup(Me.MemberID, GroupInfo.GroupId)
10:     Catch ex As Exception
11:         'Do Nothing.
12:     End Try
13: End Sub
```

Users.AddUserGroupToFolder Method

This method adds a user group to a folder and sets Permissions for the group on the folder.

C#

```
public AddUserGroupToFolder(int FolderID, int GroupID, Boolean TraverseOnly);
```

Visual Basic

```
Public Sub AddUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer, ByVal
TraverseOnly As Boolean)
```

Parameters

Parameters	Description
FolderID	The folder to which to assign the user group.
GroupID	The group ID of the user group to assign to the folder.
TraverseOnly	**True** - the group will be given Traverse permissions only. **False** - the group is given all permissions.

Remarks

This method can be used as a general-purpose method. It will work against any folder not just the ones in the StarterApp. Traverse-only is meaningful only to CMS (see page 1) user groups. Membership user groups always have traverse permissions set to True (this is hard coded in the CMS (see page 1)).

Body Source

```
 1: Public Sub AddUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer, ByVal
TraverseOnly As Boolean)
 2:     Dim UserPermissionData As New Ektron.Cms.UserPermissionData
 3:     Dim Permissions As New Ektron.Cms.API.Permissions
 4:
 5:     'Add permissions for GroupID to FolderID.
 6:     UserPermissionData.GroupId = GroupID
 7:     UserPermissionData.FolderId = FolderID
 8:
 9:     If TraverseOnly = False Then
10:         'Grant everything.
11:         UserPermissionData.CanAddToFileLib = True
12:         UserPermissionData.CanOverwriteLib = True
13:         UserPermissionData.CanAddToHyperlinkLib = True
14:         UserPermissionData.CanAddToImageLib = True
15:         UserPermissionData.CanAddToQuicklinkLib = True
16:         UserPermissionData.CanApprove = True
17:         UserPermissionData.CanBreakPending = True
18:         UserPermissionData.CanCreateTask = True
19:         UserPermissionData.CanDecline = True
20:         UserPermissionData.CanAdd = True
21:         UserPermissionData.CanDelete = True
22:         UserPermissionData.CanEdit = True
23:         UserPermissionData.CanHistory = True
24:         UserPermissionData.CanPreview = True
25:         UserPermissionData.CanPublish = True
26:         UserPermissionData.CanRestore = True
```

```
27:         UserPermissionData.CanView = True
28:         UserPermissionData.CanDeleteFolders = True
29:         UserPermissionData.CanAddFolders = True
30:         UserPermissionData.CanEditFolders = True
31:         UserPermissionData.CanEditCollections = True
32:         UserPermissionData.CanTraverseFolders = True
33:     Else
34:         'Grant only Traverse and Read-Only.
35:         UserPermissionData.IsReadOnly = False
36:         UserPermissionData.IsReadOnlyLib = False
37:         UserPermissionData.CanAddToFileLib = False
38:         UserPermissionData.CanOverwriteLib = False
39:         UserPermissionData.CanAddToHyperlinkLib = False
40:         UserPermissionData.CanAddToImageLib = False
41:         UserPermissionData.CanAddToQuicklinkLib = False
42:         UserPermissionData.CanApprove = False
43:         UserPermissionData.CanBreakPending = False
44:         UserPermissionData.CanCreateTask = False
45:         UserPermissionData.CanDecline = False
46:         UserPermissionData.CanAdd = False
47:         UserPermissionData.CanDelete = False
48:         UserPermissionData.CanEdit = False
49:         UserPermissionData.CanHistory = False
50:         UserPermissionData.CanPreview = False
51:         UserPermissionData.CanPublish = False
52:         UserPermissionData.CanRestore = False
53:         UserPermissionData.CanView = False
54:         UserPermissionData.CanDeleteFolders = False
55:         UserPermissionData.CanAddFolders = False
56:         UserPermissionData.CanEditFolders = False
57:         UserPermissionData.CanEditCollections = False
58:         UserPermissionData.CanTraverseFolders = True
59:     End If
60:
61:
62:     Try
63:         Permissions.AddItemPermission(UserPermissionData)
64:     Catch ex As Exception
65:
66:     End Try
67:
68: End Sub
```

Users.AddUserToGroup Method

This method adds a user (by user ID) to a user group (by group name).

C#

```
public AddUserToGroup(int UserID, String GroupName);
```

Visual Basic

```
Public Sub AddUserToGroup(ByVal UserID As Integer, ByVal GroupName As String)
```

Parameters

Parameters	Description
UserID	The ID of the user to add to the group.
GroupName	The name of the group to which to add the user.

Remarks

This method is more or less the same thing as AddMembertoClient (see page 54)() only this method takes a user group name whereas AddMemberToClient (see page 54)() takes a user group ID.

Body Source

```
 1: Public Sub AddUserToGroup(ByVal UserID As Integer, ByVal GroupName As String)
 2:     Dim UserGroup As New Ektron.Cms.API.User.User
 3:     Dim GroupInfo As New UserGroupData
 4:     Dim Users As New Ektron.Cms.API.User.User
 5:
 6:     GroupInfo = UserGroup.GetUserGroupByName(GroupName)
 7:     If GroupInfo IsNot Nothing Then
 8:         Try
 9:             Users.AddUserToGroup(UserID, GroupInfo.GroupId)
10:         Catch ex As Exception
11:
12:         End Try
13:     End If
14: End Sub
```

Users.RemoveMemberFromClient Method

This method removes a Membership user from a client user group.

C#

```
public RemoveMemberFromClient();
```

Visual Basic

```
Public Sub RemoveMemberFromClient()
```

Remarks

This method requires population of two Users (see page 51) properties...

1. MemberID (see page 58)
2. ClientID (see page 58)

Body Source

```
 1: Public Sub RemoveMemberFromClient()
 2:     Dim Member As New Ektron.Cms.API.User.User
 3:     Dim GroupInfo As New UserGroupData
 4:     Dim FolderAPI As New Ektron.Cms.API.Folder
 5:
 6:     GroupInfo = Member.GetUserGroupByName("starterapps.cm." &
FolderAPI.GetFolder(Me.ClientID).Name)
 7:     Try
 8:         Member.DeleteUserFromGroup(Me.MemberID, GroupInfo.GroupId, False)
 9:     Catch ex As Exception
10:         'Do Nothing.
11:     End Try
12: End Sub
```

Users.RemoveUserGroup Method

This method removes a user group (by user group name).

C#

```
public RemoveUserGroup(String GroupName);
```

Visual Basic

```
Public Sub RemoveUserGroup(ByVal GroupName As String)
```

Parameters

Parameters	Description
GroupName	The name of the user group to remove.

Remarks

This method is called by RemoveClient() method. When a client is removed, so is its Membership user group.

Body Source

```
 1: Public Sub RemoveUserGroup(ByVal GroupName As String)
 2:     Dim UserGroup As New API.User.User
 3:     Dim UserGroupData As New UserGroupData
 4:     Try
 5:         UserGroupData = UserGroup.GetUserGroupByName(GroupName)
 6:         UserGroup.DeleteUserGroup(UserGroupData.GroupId)
 7:     Catch ex As Exception
 8:
 9:     End Try
10: End Sub
```

Users Properties

The properties of the Users class are listed here.

Public Properties

	Name	Description
	ClientID (see page 58)	Holds the ID of the selected client.
	MemberID (see page 58)	The MemberID property is used to hold the ID of a membership user created at run-time via Registration widgets.

Legend

	Property

Users.ClientID Property

Holds the ID of the selected client.

C#

```
public int ClientID;
```

Visual Basic

```
Public Property ClientID() As Integer
```

Returns

The selected client's ID.

Users.MemberID Property

The MemberID property is used to hold the ID of a membership user created at run-time via Registration widgets.

C#

```
public int MemberID;
```

Visual Basic

```
Public Property MemberID() As Integer
```

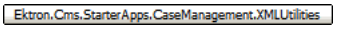
Returns

The ID of the membership user created at run-time.

XMLUtilities Class

Contains methods and properties for handling the XML used within the Case Manager Starter Application.

Class Hierarchy



C#

```
public class XMLUtilities;
```

Visual Basic

```
Public Class XMLUtilities
```

File

CaseManagement.vb

XMLUtilities Enumerations

The enumerations of the XMLUtilities class are listed here.

Enumerations

	Name	Description
	XMLSourceTypes (see page 59)	Sets the input type of the XML to be transformed.
	XSLTSourceTypes (see page 59)	Sets the input type of the XSLT to be used as the transforming document.

Legend

	Enumeration

Ektron.Cms.StarterApps.CaseManagement.XMLUtilities.XMLSourceTypes Enumeration

Sets the input type of the XML to be transformed.

C#

```
public enum XMLSourceTypes {
}
```

Visual Basic

```
Public Enum XMLSourceTypes
End Enum
```

File

CaseManagement.vb

Remarks

If the type is set to XMLDocument (see page 64), you must set the XMLDocument (see page 64) property to the XMLDocument (see page 64) you wish to transform. Otherwise, set the the XMLSource (see page 64) property to the string location of the XMLDocument (see page 64) you wish to transform.

Ektron.Cms.StarterApps.CaseManagement.XMLUtilities.XSLTSourceTypes Enumeration

Sets the input type of the XSLT to be used as the transforming document.

C#

```
public enum XSLTSourceTypes {
}
```

Visual Basic

```
Public Enum XSLTSourceTypes
```

```
End Enum
```

File

CaseManagement.vb

Remarks

You must also set the the XSLTSource (see page 65) property to the string location of the XSLTDocument you wish to use as the transformation document.

XMLUtilities Methods

The methods of the XMLUtilities class are listed here.

Public Methods

	Name	Description
	ApplyXMLAttribute (see page 60)	This method applies XML attributes to an XML Node.
	PrettyPrintXML (see page 60)	Returns the contents of XMLDocument (see page 64) nicely indented.
	TransformXML (see page 61)	This method is a general purpose XML transformation method. This method allows arguments to be passed to the XSLT transformation.

Legend

	Method

XMLUtilities.ApplyXMLAttribute Method

This method applies XML attributes to an XML Node.

C#

```
public ApplyXMLAttribute(XmlDocument IADoc, XmlNode Node, String AttributeName, String
AttributeValue);
```

Visual Basic

```
Public Sub ApplyXMLAttribute(ByRef IADoc As XmlDocument, ByRef Node As XmlNode, ByVal
AttributeName As String, ByVal AttributeValue As String)
```

Parameters

Parameters	Description
IADoc	The XML document being parsed.
Node	The node in which to add the attribute.
AttributeName	The attribute's name.
AttributeValue	The attribute's value.

Remarks

This is a general purpose method meant to simplify adding attributes to nodes. It's meant to help keep code tidy.

Body Source

```
1: Public Sub ApplyXMLAttribute(ByRef IADoc As XmlDocument, ByRef Node As XmlNode, ByVal
AttributeName As String, ByVal AttributeValue As String)
2:     Dim Attribute As XmlAttribute
3:     Attribute = IADoc.CreateAttribute(AttributeName)
4:     Attribute.InnerText = AttributeValue
5:     Node.Attributes.Append(Attribute)
6: End Sub
```

XMLUtilities.PrettyPrintXML Method

Returns the contents of XMLDocument (see page 64) nicely indented.

C#

```
public String PrettyPrintXML(System.Xml.XmlDocument XMLDocument);
```

Visual Basic

```
Public Function PrettyPrintXML(ByRef XMLDocument As System.Xml.XmlDocument) As String
```

Parameters

Parameters	Description
XMLDocument	The XMLdocument to format.

Returns

Formatted string.

Remarks

Used primarily to make long XML/XHTML readable.

Body Source

```
 1: Public Function PrettyPrintXML(ByRef XMLDocument As System.Xml.XmlDocument) As String
 2:     'Create stream in which to load the formatted XML document.
 3:     Dim OutputStream As System.IO.Stream = New System.IO.MemoryStream
 4:
 5:     'Create XMLTextWriter with formatting specifics, and direct output into OutputStream.
 6:     Dim MyXmlTextWriter As New System.Xml.XmlTextWriter(OutputStream, Nothing)
 7:     MyXmlTextWriter.Formatting = Formatting.Indented
 8:     MyXmlTextWriter.Indentation = 5
 9:     MyXmlTextWriter.IndentChar = " "
10:
11:     'Load the contents of XMLDocument argument into XMLTextWriter.
12:     XMLDocument.Save(MyXmlTextWriter)
13:
14:     'Load Formatted XML Stream into StreamReader.
15:     Dim XMLStreamReader As New System.IO.StreamReader(OutputStream)
16:     OutputStream.Flush()
17:     OutputStream.Position = 0
18:
19:     'Return Indented XML String.
20:     PrettyPrintXML = XMLStreamReader.ReadToEnd()
21:     XMLStreamReader.Close()
22: End Function
```

XMLUtilities.TransformXML Method

This method is a general purpose XML transformation method. This method allows arguments to be passed to the XSLT transformation.

C#

```
public String TransformXML();
```

Visual Basic

```
Public Function TransformXML() As String
```

Returns

XML transformed by an XSLT stylesheet.

Remarks

This method takes XML source from

1. A string of xml text.
2. A URI (http://something).

3. A file on the file system.
4. An object of type system.xml.xmldocument.

To take advantage of the first three XML source type options, you must:

a. Set the XMLSourceType (see page 65) property to XMLString, URI, or File.

b. Set the XMLSource (see page 64) property to the string containing the corresponding value.

To take advantage of the fourth XML source type, you must:

a. Set the XMLSourceType (see page 65) property to XMLDocument (see page 64).

b. Set the XMLDocument (see page 64) property to the XML document you wish to transform.

This method takes XSLT source from

1. A string of xml text.
2. A URI (http://something).
3. A file on the file system.

To set the XSLT source type options, you must:

a. Set the XSLTSourceType (see page 65) property to XMLString, URI, or File.

b. Set the XMLSource (see page 64) property to the string containing the corresponding value.

This method takes an arbitrary number of paramaters to pass into the XSLT transformation. In order to pass params into the XSLT transform, you must:

a. Populate a NameValueCollection (param name, param, value).

b. Set the XSLTParams (see page 65) property to your populated NameValueCollection.

Body Source

```
 1: Public Function TransformXML() As String
 2:     Dim myXML As New System.Xml.XmlDocument
 3:     Select Case Me.XMLSourceTypeValue
 4:         Case XMLSourceTypes.File
 5:             Try
 6:                 myXML.Load(System.Web.HttpContext.Current.Server.MapPath(XMLSourceValue))
 7:             Catch ex As Exception
 8:                 Return "Cannot Load XML File: " & Me.XMLSource
 9:             End Try
10:         Case XMLSourceTypes.URI
11:             Try
12:                 myXML.Load(XMLSourceValue)
13:             Catch ex As Exception
14:                 Return "Cannot Load XML File: " & Me.XMLSource
15:             End Try
16:         Case XMLSourceTypes.XMLString
17:             Try
18:                 myXML.LoadXml(XMLSourceValue)
19:             Catch ex As Exception
20:                 Return "Cannot Load XML String: " & ex.Message
21:             End Try
22:         Case XMLSourceTypes.XMLDocument
23:             Try
24:                 myXML = Me.XMLDocument
```

```
25:             Catch ex As Exception
26:                 Return "Cannot Load XML Document: " & ex.Message
27:             End Try
28:         Case Else
29:             Try
30:                 Throw New Exception("XML Type Property (File, URI, XMLString, XMLDocument)
Not Set")
31:             Catch ex As Exception
32:                 Return "Cannot Load XML: " & ex.Message
33:             End Try
34:     End Select
35:
36:     Dim myXSLT As New System.Xml.Xsl.XslCompiledTransform
37:     Select Case Me.XSLTSourceTypeValue
38:         Case XSLTSourceTypes.File
39:             Try
40:                 myXSLT.Load(System.Web.HttpContext.Current.Server.MapPath(XSLTSourceValue))
41:             Catch ex As Exception
42:                 Return "Cannot Load XSLT File: " & Me.XSLTSource
43:             End Try
44:         Case XSLTSourceTypes.URI
45:             Try
46:                 myXSLT.Load(XSLTSourceValue)
47:             Catch ex As Exception
48:                 Return "Cannot Load XSLT File: " & Me.XSLTSource
49:             End Try
50:         Case XSLTSourceTypes.XSLTString
51:             Try
52:                 Dim myXSLTString As New XmlDocument()
53:                 myXSLTString.LoadXml(XSLTSourceValue)
54:                 myXSLT.Load(myXSLTString)
55:             Catch ex As Exception
56:                 Return "Cannot Load XSLT String: " & ex.Message
57:             End Try
58:         Case Else
59:             Try
60:                 Throw New Exception("XSLT Type Property (File, URI, XSLTString) Not Set")
61:             Catch ex As Exception
62:                 Return "Cannot Load XSLT: " & ex.Message
63:             End Try
64:     End Select
65:
66:     Dim myOutputStream As New System.IO.MemoryStream()
67:     Dim myXSLTArgs As New System.Xml.Xsl.XsltArgumentList
68:
69:     If Not myXSLTParams Is Nothing Then
70:         Dim i As Integer
71:         For i = 0 To Me.XSLTParams.Count - 1
72:             myXSLTArgs.AddParam(Me.XSLTParams.GetKey(i), "", Me.XSLTParams.GetValues(i)(0))
73:         Next
74:         Try
75:             myXSLT.Transform(myXML, myXSLTArgs, myOutputStream)
76:         Catch ex As Exception
77:             Return ex.Message
78:         End Try
79:     Else
80:         Try
81:             myXSLT.Transform(myXML, Nothing, myOutputStream)
82:         Catch ex As Exception
83:             Return ex.Message
84:         End Try
85:     End If
86:
87:     myOutputStream.Flush()
88:     myOutputStream.Position = 0
89:
```

```
90:     Dim myXMLStreamReader As New System.IO.StreamReader(myOutputStream)
91:     Dim myTransformedXML As String
92:
93:     myTransformedXML = myXMLStreamReader.ReadToEnd()
94:     myXMLStreamReader.Close()
95:
96:     Return myTransformedXML
97:
98: End Function
```

XMLUtilities Properties

The properties of the XMLUtilities class are listed here.

Public Properties

	Name	Description
	XMLDocument (see page 64)	Holds the XMLDocument to be transformed.
	XMLSource (see page 64)	If XMLSourceType (see page 65) is NOT set to XMLDocument (see page 64), this property holds the location of the XMLDocument (see page 64) to be transformed.
	XMLSourceType (see page 65)	Holds the type of input XML document.
	XSLTParams (see page 65)	Specifies any parameters to be passed into the XSLT transformation.
	XSLTSource (see page 65)	Specifies where the XSLT document is located.
	XSLTSourceType (see page 65)	Sets the type of document to be used as transformation source - specifies where this document lives.

Legend

	Property

XMLUtilities.XMLDocument Property

Holds the XMLDocument to be transformed.

C#

```
public XmlDocument XMLDocument;
```

Visual Basic

```
Public Property XMLDocument() As XmlDocument
```

Returns

XmlDocument

Remarks

Only needs to be populated if the XMLSourceType (see page 65) is set to XmlDocument.

XMLUtilities.XMLSource Property

If XMLSourceType (see page 65) is NOT set to XMLDocument (see page 64), this property holds the location of the XMLDocument (see page 64) to be transformed.

C#

```
public String XMLSource;
```

Visual Basic

```
Public Property XMLSource() As String
```

Returns

The location of the XML Document to be transformed.

Remarks

Only needs to be populated if the XMLSourceType (see page 65) is not set to XMLDocument (see page 64).

XMLUtilities.XMLSourceType Property

Holds the type of input XML document.

C#

```
public XMLSourceTypes XMLSourceType;
```

Visual Basic

```
Public Property XMLSourceType() As XMLSourceTypes
```

Returns

XML Document source type - XMLString, URI, File, or XMLDocument (see page 64).

Remarks

If the type is set to XMLDocument (see page 64), you must set the XMLDocument (see page 64) property to the XMLDocument (see page 64) you wish to transform. Otherwise, set the the XMLSource (see page 64) property to the string location of the XMLDocument (see page 64) you wish to transform.

XMLUtilities.XSLTParams Property

Specifies any parameters to be passed into the XSLT transformation.

C#

```
public NameValueCollection XSLTParams;
```

Visual Basic

```
Public Property XSLTParams() As NameValueCollection
```

Returns

List of parameters to be passed into the transform

Remarks

Optional - can hold any number of parameters. Specified as "param name", "value." You must define these parameters in your XSLT file to use them.

XMLUtilities.XSLTSource Property

Specifies where the XSLT document is located.

C#

```
public String XSLTSource;
```

Visual Basic

```
Public Property XSLTSource() As String
```

Returns

Location of the XSLT document.

XMLUtilities.XSLTSourceType Property

Sets the type of document to be used as transformation source - specifies where this document lives.

C#

```
public XSLTSourceTypes XSLTSourceType;
```

Visual Basic

```
Public Property XSLTSourceType() As XSLTSourceTypes
```

Returns

String

Remarks

Specifies how to load the XSLT document to be used as transformation source.

Ektron.Cms.StarterApps.ProjectManagement Namespace

This is namespace Ektron.Cms.StarterApps.ProjectManagement.

Classes

	Name	Description
	Clients (see page 66)	Contains methods for managing clients in the Project Manager Starter Application.
	ProjectManagement (see page 71)	Contains methods and properties for the overall creation and management of the Project Management application. Important methods worth highlighting are **Initialize (see page 83)** and **GetInformationArchitecture (see page 76).** Initialize (see page 83) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 76) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	Projects (see page 94)	Contains methods and properties that manage the project aspect of the project management site. This includes functionality for Blogs, Discussion Boards, Taxonomy, Wiki (see page 130) and Folders.
	Users (see page 115)	Contains methods for managing users in the Project Manager Starter Application.
	XMLUtilities (see page 122)	Contains methods and properties for handling the XML used within the Project Manager Starter Application.

Legend

	Class

Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	Clients (see page 66)	Contains methods for managing clients in the Project Manager Starter Application.
	ProjectManagement (see page 71)	Contains methods and properties for the overall creation and management of the Project Management application. Important methods worth highlighting are **Initialize (see page 83)** and **GetInformationArchitecture (see page 76).** Initialize (see page 83) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 76) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	Projects (see page 94)	Contains methods and properties that manage the project aspect of the project management site. This includes functionality for Blogs, Discussion Boards, Taxonomy, Wiki (see page 130) and Folders.
	Users (see page 115)	Contains methods for managing users in the Project Manager Starter Application.
	XMLUtilities (see page 122)	Contains methods and properties for handling the XML used within the Project Manager Starter Application.

Legend

	Class

Clients Class

Contains methods for managing clients in the Project Manager Starter Application.

Class Hierarchy

C#

```
public class Clients;
```

Visual Basic

```
Public Class Clients
```

File

ProjectManagement.vb

Clients Methods

The methods of the Clients class are listed here.

Public Methods

	Name	Description
	AddClient (see page 67)	This method adds a new client to the Project Management. StarterApp.
	GetAllClients (see page 69)	This method gathers all the client names and client folder ID's that the current user has permissions to view.
	GetClientXMLForInformationArchitecture (see page 69)	This method gathers the Information Architecture for a particular client.
	RemoveClient (see page 70)	This method removes a client from the Project Management StarterApp.

Legend

	Method

Clients.AddClient Method

This method adds a new client to the Project Management. StarterApp.

C#

```
public int AddClient(String ClientName);
```

Visual Basic

```
Public Function AddClient(ByVal ClientName As String) As Integer
```

Parameters

Parameters	Description
`ClientName`	The name of the client to add.

Returns

The ID of the newly created Client folder.

Remarks

This method does several things besides creating a CMS (see page 1) folder for a client.

1. If the client name already exists it doesn't do anything.
2. It breaks both content and metadata inheritance on the folder - this is so we can set the exact permissions and metadata models we wish.
3. It sets the folder to PRIVATE.
4. It creates a new Membership user group ("Starterapps.pm.~client name~").
5. It adds the "starterapps.pm" CMS (see page 1) user group to the folder.

Body Source

```
 1: Public Function AddClient(ByVal ClientName As String) As Integer
 2:     Dim PM As New Ektron.Cms.StarterApps.ProjectManagement.ProjectManagement
 3:     Dim PMFolderID As Integer
 4:     Dim ClientList As New NameValueCollection
 5:     Dim ClientExists As Boolean = False
 6:     Dim ClientFolderID As Integer
 7:     Dim ClientFolder As New FolderRequest
 8:     Dim Projects As New Projects
 9:     Dim Users As New Ektron.Cms.StarterApps.ProjectManagement.Users
10:     Dim MembershipUserGroupID As Integer
11:     Dim FolderAPI As New Ektron.Cms.API.Folder
12:
13:     'Unescape Client Name.
14:     ClientName = System.Web.HttpUtility.HtmlDecode(ClientName)
15:
16:     'Get the Project Management folder ID.
17:     PMFolderID = PM.Initialize()
18:
19:     'Get all client folders inside the Project Management Folder.
20:     ClientList = GetAllClients()
21:
22:     'Loop to see if the client folder already exists.
23:     If Not ClientList.Count = 0 Then
24:         Dim i As Integer
25:         For i = 0 To ClientList.Count - 1
26:             If System.Web.HttpUtility.HtmlDecode(ClientList.GetKey(i)) = ClientName Then
27:                 ClientExists = True
28:                 ClientFolderID = ClientList.GetValues(i)(0)
29:                 Exit For
30:             End If
31:         Next
32:     End If
33:
34:     'If client does not exist, create the client folder.
35:     If ClientExists = False Then
36:
37:         'Create Client Folder.
38:         ClientFolder.FolderName = ClientName
39:         ClientFolder.ParentId = PMFolderID
40:         ClientFolder.FolderDescription = ClientName & " Project Management Folder"
41:         ClientFolder.SiteMapPathInherit = False
42:         ClientFolder.MetaInherited = 0
43:         FolderAPI.AddCommunityFolder(ClientFolder)
44:
45:         'Get the client ID.
46:         ClientFolderID = ClientFolder.FolderId
47:         'Break inheritance.
48:         PM.SetFolderPermissions(ClientFolderID)
49:         'Make the Folder Private.
50:         PM.SetFolderPrivate(ClientFolderID)
51:         'Add MembershipUserGroup.
52:         MembershipUserGroupID = Users.AddMembershipUserGroup("Starterapps.pm." &
ClientName)
53:         'Add MembershipUserGroup To Client Folder.
54:         If MembershipUserGroupID <> -1 Then
55:             Users.AddMembershipUserGroupToFolder(ClientFolderID, MembershipUserGroupID)
56:         End If
57:     Else
58:         ClientFolderID = -2
59:     End If
60:     'Return the ID of the client folder.
61:     Return ClientFolderID
62: End Function
```

Clients.GetAllClients Method

This method gathers all the client names and client folder ID's that the current user has permissions to view.

C#

```
public NameValueCollection GetAllClients();
```

Visual Basic

```
Public Function GetAllClients() As NameValueCollection
```

Returns

NameValueCollection - client name, client folder ID.

Remarks

This method uses GetChildFolders and therefore respects permissions.

Body Source

```
 1: Public Function GetAllClients() As NameValueCollection
 2:     Dim PM As New Ektron.Cms.StarterApps.ProjectManagement.ProjectManagement
 3:     Dim PMFolderID As Integer
 4:     Dim PMFolderData() As FolderData
 5:     Dim PMFolderItem As New FolderData
 6:     Dim Clients As New NameValueCollection
 7:     Dim FolderAPI As New Ektron.Cms.API.Folder
 8:
 9:     PMFolderID = PM.Initialize()
10:
11:     'Create nameValueCollection of folder names and folder IDs.
12:     PMFolderData = FolderAPI.GetChildFolders(PMFolderID, False)
13:     If Not PMFolderData Is Nothing Then
14:         For Each PMFolderItem In PMFolderData
15:             Clients.Add(PMFolderItem.Name, PMFolderItem.Id)
16:         Next
17:     End If
18:
19:     Return Clients
20: End Function
```

Clients.GetClientXMLForInformationArchitecture Method

This method gathers the Information Architecture for a particular client.

C#

```
public System.Xml.XmlDocument GetClientXMLForInformationArchitecture(int ClientID);
```

Visual Basic

```
Public Function GetClientXMLForInformationArchitecture(ByVal ClientID As Integer) As
System.Xml.XmlDocument
```

Parameters

Parameters	Description
ClientID	The ID of the client (folder ID) for which the IA is to be generated.

Returns

An XML document containing the Information Architecture for a particular client.

Remarks

This method is a more efficient version of Project.GetInformationArchitectureXML(), which returns the entire IA tree for a particular user. This method restricts the tree to a particular branch (one client), so it's a bit more efficient. Project.GetInformationArchitect() also has a boolean switch to include all the members of the client's Membership user group.

This method does not have such a switch.

Body Source

```
 1: Public Function GetClientXMLForInformationArchitecture(ByVal ClientID As Integer) As
System.Xml.XmlDocument
 2:
 3:     Dim PM As New Ektron.Cms.StarterApps.ProjectManagement.ProjectManagement
 4:     Dim DocumentNode, ClientNode As System.Xml.XmlNode
 5:     Dim XMLUtils As New Ektron.Cms.StarterApps.ProjectManagement.XMLUtilities
 6:     Dim FolderAPI As New Ektron.Cms.API.Folder
 7:     Dim ClientFolderData As New Ektron.Cms.FolderData
 8:     Dim FolderData As New Ektron.Cms.FolderData
 9:
10:     GetClientXMLForInformationArchitecture = New System.Xml.XmlDocument
11:
12:     FolderData = FolderAPI.GetFolder(ClientID)
13:
14:     'Initalize the IA XML document.
15:     DocumentNode = GetClientXMLForInformationArchitecture.CreateElement("ektron")
16:     GetClientXMLForInformationArchitecture.AppendChild(DocumentNode)
17:
18:     ClientNode = GetClientXMLForInformationArchitecture.CreateElement("li")
19:     XMLUtils.ApplyXMLAttribute(GetClientXMLForInformationArchitecture, ClientNode, "id",
ClientID)
20:     XMLUtils.ApplyXMLAttribute(GetClientXMLForInformationArchitecture, ClientNode,
"label", FolderData.Name)
21:     XMLUtils.ApplyXMLAttribute(GetClientXMLForInformationArchitecture, ClientNode,
"class", "Client")
22:     GetClientXMLForInformationArchitecture.DocumentElement.AppendChild(ClientNode)
23:
24:     Return GetClientXMLForInformationArchitecture
25: End Function
```

Clients.RemoveClient Method

This method removes a client from the Project Management StarterApp.

C#

```
public RemoveClient(int ClientFolderID);
```

Visual Basic

```
Public Sub RemoveClient(ByVal ClientFolderID As Integer)
```

Parameters

Parameters	Description
ClientFolderID	The ID of the client (folder ID) to be removed.

Remarks

1. This method calls Projects.RemoveProject (see page 112)() to individually remove each project to which this client is a parent.project.
2. This method removes the client Membership group "pm.~client name~."
3. This method removes the client folder.

Body Source

```
 1: Public Sub RemoveClient(ByVal ClientFolderID As Integer)
 2:     Dim UserGroup As New Ektron.Cms.StarterApps.ProjectManagement.Users
 3:     Dim Projects As New Ektron.Cms.StarterApps.ProjectManagement.Projects
 4:     Dim ProjectFolders() As Ektron.Cms.FolderData
 5:     Dim ProjectFolderID As Integer
 6:     Dim FolderAPI As New Ektron.Cms.API.Folder
 7:
 8:     'Delete all of the client's projects
```

```
 9:     ProjectFolders = FolderAPI.GetChildFolders(ClientFolderID)
10:     If ProjectFolders IsNot Nothing Then
11:         For Each ProjectFolder As FolderData In ProjectFolders
12:             ProjectFolderID = ProjectFolder.Id
13:             Try
14:                 Projects.RemoveProject(ProjectFolderID)
15:             Catch ex As Exception
16:
17:             End Try
18:
19:         Next
20:     End If
21:
22:
23:     'Delete client Membership user group.
24:     Try
25:         UserGroup.RemoveUserGroup("Starterapps.pm." &
FolderAPI.GetFolder(ClientFolderID).Name)
26:     Catch ex As Exception
27:
28:     End Try
29:
30:
31:     'Delete client folder.
32:     Try
33:         FolderAPI.DeleteFolderById(ClientFolderID)
34:     Catch ex As Exception
35:
36:     End Try
37:
38:
39:
40: End Sub
```

ProjectManagement Class

Contains methods and properties for the overall creation and management of the Project Management application. Important methods worth highlighting are **Initialize (see page 83)** and **GetInformationArchitecture (see page 76).** Initialize (see page 83) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 76) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.

Class Hierarchy

Ektron.Cms.StarterApps.ProjectManagement.ProjectManagement

C#

```
public class ProjectManagement;
```

Visual Basic

```
Public Class ProjectManagement
```

File

ProjectManagement.vb

ProjectManagement Methods

The methods of the ProjectManagement class are listed here.

Private Methods

	Name	Description
	AddMetadataDefinition (see page 73)	This method is called by Initialize (see page 83)() when creating the required metadata values
	GetBlogPermissionsType (see page 74)	This method gets the logical "type" of blog.
	GetMembersXML (see page 78)	This method inserts member-information to the Client nodes in the InformationArchitecture.xml generated by GetInformationArchitectureXML (see page 76)().
	GetProjectXML (see page 81)	This method collects blog component information and inserts this information into InformationArchitecture.xml. This method recursively calls itself to gather all folders/sub-folders of a particular project.
	MembershipUserTraverseOnly (see page 86)	This method checks to see if a Membership user ID has been specifically granted permissions to a folder.

Legend

	Method
	Private

Public Methods

	Name	Description
	AddFolder (see page 72)	Creates StarterApps (see page 1) and Project Management folders.
	GetBreadcrumbs (see page 75)	This method gets the breadcrumb trail based on the user's navigation selection.
	GetInformationArchitecture (see page 76)	This method gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	GetInformationArchitectureXML (see page 76)	This method generates an XML-based hierarchy of clients, projects and project components to which the user has permissions.
	GetIssuesSmartFormID (see page 78)	This method returns the ID of the Milestones SmartForm.
	GetMetadataDefinitionID (see page 80)	This method gets IDs for any of the three metadata definitions for the Project Management StarterApp.
	GetSmartFormID (see page 83)	This method returns the ID of the Milestones SmartForm.
	Initialize (see page 83)	The Initialize() method is executed upon each page load to ensure StartApp has the components it needs, and returns the "Project Management" folder ID.
	RegisterMilestonesSmartForm (see page 87)	Imports the Milestones SmartFrom from "xml/MilestonesSmartForm.xml"
	RegisterTemplate (see page 88)	This method registers our templates inside CMS400 during Project Management StarterApp Initialization.
	RemoveInheritedUserGroups (see page 89)	This method removes any user groups and users assigned to a folder.
	SetFolderPermissions (see page 90)	This method breaks content and metadata inheritance for folders. You may specify which inheritance property you wish to break or specify if you wish to break both.
	SetFolderPrivate (see page 90)	This method sets the folder to private status.
	TransformInformationArchitectureXML (see page 91)	This method transforms Information Architecture XML to an XHTML unordered list.

Legend

	Method

ProjectManagement.AddFolder Method

Creates StarterApps (see page 1) and Project Management folders.

C#

```
public int AddFolder(String FolderName, int ParentID);
```

Visual Basic

```
Public Function AddFolder(ByVal FolderName As String, ByVal ParentID As Integer) As Integer
```

Parameters

Parameters	Description
FolderName	This should either be "StarterApps (see page 1)" or "Project Management."
ParentID	This should either by "0" if it's the "StarterApps (see page 1)" folder or StarterAppsFolderID if it's the "Project Management" folder.

Returns

Folder ID

Remarks

This method is called by the Intialize() method.

Body Source

```
 1: Public Function AddFolder(ByVal FolderName As String, ByVal ParentID As Integer) As Integer
 2:
 3:     Dim FolderRequest As New FolderRequest
 4:     Dim FolderDataItem As New FolderData
 5:
 6:     FolderRequest.FolderName = FolderName
 7:     FolderRequest.ParentId = ParentID
 8:     FolderRequest.MetaInherited = 0
 9:     FolderRequest.StyleSheet = "NoCss.css"
10:     FolderRequest.TemplateFileName = "NoTemplate.aspx"
11:     FolderRequest.TaxonomyInherited = False
12:     FolderRequest.CategoryRequired = False
13:     FolderRequest.FolderType = Ektron.Cms.Common.EkEnumeration.FolderType.Content
14:     FolderRequest.suppressNotification = True
15:     FolderRequest.FolderDescription = "Ektron Starter Apps Folder"
16:     FolderRequest.SiteMapPathInherit = False
17:     FolderAPI.AddFolder(FolderRequest)
18:
19:     Return FolderRequest.FolderId
20: End Function
```

ProjectManagement.AddMetadataDefinition Method

This method is called by Initialize (see page 83)() when creating the required metadata values

C#

```
private int AddMetadataDefinition(MetadataDefinitionTypes Type);
```

Visual Basic

```
Private Function AddMetadataDefinition(ByVal Type As MetadataDefinitionTypes) As Integer
```

Parameters

Parameters	Description
Type	Values: "Starterapps.pm.DefaultContent", "Starterapps.pm.MembershipBlog", "Starterapps.pm.CMSBlog"

Returns

Metadata definition ID

Remarks

- **"pm.DefaultContent"** - used to identify a content entry to load by default when the Wiki (see page 130) loads without any user-selected content.
- **"pm.MembershipBlog"** - used during the creation of InformationArchitecture.xml to identify the audience of the blog ("Membership Blogs" are actually viewable by both CMS (see page 1) users AND Membership users.)
- **"pm.CMSBlog"** - used during the creation of InformationArchitecture.xml to identify the audience of the blog ("CMS (see page 1) Blogs" are viewable only by CMS (see page 1) users.)

Body Source

```
 1: Private Function AddMetadataDefinition(ByVal Type As MetadataDefinitionTypes) As Integer
 2:     Dim Metadata As New Metadata
 3:     Dim MetadataCollection As New Ektron.Cms.ContentMetaData
 4:     Dim MetadataTypeName As String
 5:
 6:     Select Case Type
 7:         Case MetadataDefinitionTypes.wiki
 8:             MetadataTypeName = "starterapps.pm.defaultcontent"
 9:         Case MetadataDefinitionTypes.membershipblog
10:             MetadataTypeName = "starterapps.pm.membershipblog"
11:         Case MetadataDefinitionTypes.cmsblog
12:             MetadataTypeName = "starterapps.pm.cmsblog"
13:         Case Else
14:             Exit Function
15:     End Select
16:
17:     MetadataCollection.TypeName = MetadataTypeName
18:     MetadataCollection.DefaultText = "No"
19:
20:     'Title Values:
21:     'text, number, byte, double, float, integer, long, short, date, boolean, select1
(select from a list), select (multiple selections).
22:     MetadataCollection.Title = "boolean"
23:     'TagType Values:
24:     '100 - Searchable Property
25:     '1 - Meta
26:     '0 - HTML
27:     '2 - Collection
28:     '3 - List Summary
29:     '4 - Content
30:     '5 - Image
31:     '7 - File
32:     MetadataCollection.TagType = "100"
33:     MetadataCollection.Editable = True
34:     MetadataCollection.Separater = ";"
35:     MetadataCollection.SearchAllowed = True
36:     MetadataCollection.SelectableText = "No;Yes"
37:     MetadataCollection.MetaDisplayEE = False
38:     MetadataCollection.MetaDisplay = False
39:     MetadataCollection.IsSelectableOnly = True
40:
41:     Try
42:         Metadata.AddMetaDataType(MetadataCollection)
43:     Catch ex As Exception
44:         Return -1
45:     End Try
46:
47:     Return MetadataCollection.TypeId
48:
49: End Function
```

ProjectManagement.GetBlogPermissionsType Method

This method gets the logical "type" of blog.

C#

```
private String GetBlogPermissionsType(int BlogID);
```

Visual Basic

```
Private Function GetBlogPermissionsType(ByVal BlogID As Integer) As String
```

Parameters

Parameters	Description
BlogID	ID of a blog inside the Project Management StarterApp folder tree.

Returns

String: either "MembershipBlog", or "CMSBlog."

Remarks

This value is based on a metadata setting assigned at the time the blog was created. This value differentiates blogs intended for CMS (see page 1) users from those intended for CMS (see page 1) AND Membership users.

Body Source

```
 1: Private Function GetBlogPermissionsType(ByVal BlogID As Integer) As String
 2:     Dim Metadata As New Ektron.Cms.API.CustomFields
 3:     Dim MetadataCollection As New Collection
 4:     Dim MetadataType As String = "None"
 5:     Dim BlogTypeArray(0) As String
 6:     Dim BlogType As String
 7:
 8:     MetadataCollection = Metadata.GetFieldsByFolder(BlogID, 1033)
 9:     For Each MetadataItem As Collection In MetadataCollection
10:         If MetadataItem IsNot Nothing Then
11:             If MetadataItem.Item("Assigned") <> 0 Then
12:                 MetadataType = MetadataItem.Item("CustomFieldName")
13:             End If
14:         End If
15:     Next
16:     BlogTypeArray = Split(MetadataType, ".")
17:     BlogType = BlogTypeArray(BlogTypeArray.Length - 1)
18:     Return BlogType
19: End Function
```

ProjectManagement.GetBreadcrumbs Method

This method gets the breadcrumb trail based on the user's navigation selection.

C#

```
public String GetBreadcrumbs();
```

Visual Basic

```
Public Function GetBreadcrumbs() As String
```

Returns

An XHTML unordered list.

Remarks

To generate the breadcrumb trail, this method transforms InformationArchitecture.xml with Breadcrumbs.xslt.

Body Source

```
 1: Public Function GetBreadcrumbs() As String
 2:     Dim XMLUtils As New XMLUtilities
 3:
 4:     'Set XML Args.
 5:     XMLUtils.XMLSourceType = XMLUtilities.XMLSourceTypes.XMLDocument
 6:     XMLUtils.XMLDocument = InformationArchitectureXML
 7:     'Set XSLT Params.
 8:     XMLUtils.XSLTParams.Add("ProjectID", Me.ProjectID)
 9:     XMLUtils.XSLTParams.Add("Template", Me.Template)
10:     XMLUtils.XSLTParams.Add("TemplateLabel", Me.TemplateLabel)
11:     'Set XSLT Args.
```

```
12:     XMLUtils.XSLTSourceType = XMLUtilities.XSLTSourceTypes.File
13:     XMLUtils.XSLTSource = Me.IAPath & "xml/Breadcrumbs.xslt"
14:     'Transform Information Architectre.
15:     GetBreadcrumbs = XMLUtils.TransformXML()
16: End Function
```

ProjectManagement.GetInformationArchitecture Method

This method gets the information architecture (navigation/content hierarchy) the current user has permissions to access.

C#

```
public String GetInformationArchitecture();
```

Visual Basic

```
Public Function GetInformationArchitecture() As String
```

Returns

XHTML unordered list representing the clients and projects with which the current user has permissions to view/interact.

Remarks

This method crawls the Project Management folder tree looking for branches to which the user has permissions to access.

Steps:

1. Represent this structure as XML.
2. Pass this XML to InformationArchitecture.xslt.
3. Return the transformed XML as XHTML unordered list.

Body Source

```
 1: Public Function GetInformationArchitecture() As String
 2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
 3:     Dim InformationArchitectureXHTML As String = ""
 4:     'Get Information Architecture.
 5:
 6:     If Me.InformationArchitectureXML Is Nothing Then
 7:         InformationArchitectureXML = GetInformationArchitectureXML(False)
 8:     Else
 9:         InformationArchitectureXML = Me.InformationArchitectureXML
10:     End If
11:     'Transofrm Information Architecture XML to XHTML.
12:     If (InformationArchitectureXML IsNot Nothing) Then
13:         InformationArchitectureXHTML = TransformInformationArchitectureXML()
14:     End If
15:     Return InformationArchitectureXHTML
16:
17: End Function
```

ProjectManagement.GetInformationArchitectureXML Method

This method generates an XML-based hierarchy of clients, projects and project components to which the user has permissions.

C#

```
public System.Xml.XmlDocument GetInformationArchitectureXML(Boolean IncludeMembers);
```

Visual Basic

```
Public Function GetInformationArchitectureXML(ByVal IncludeMembers As Boolean) As
System.Xml.XmlDocument
```

Returns

An XML hierarchy of clients, projects and project components.

Remarks

This hierarchy is generated once per-page load.

This hierarchy contains all information necessary to create navigation and gather context for each request.

This accomplishes three main things...

1. All permissions checks are performed once.
2. All folder and content IDs are looked-up once.
3. Context information is made available for each URI.

Body Source

```
 1: Public Function GetInformationArchitectureXML(ByVal IncludeMembers As Boolean) As
System.Xml.XmlDocument
 2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
 3:     Dim PMfolderID As Integer
 4:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 5:     Dim FolderData() As FolderData = Nothing
 6:     Dim DocumentNode, ErrorNode, CurrentNode As System.Xml.XmlNode
 7:     Dim XMLUtils As New XMLUtilities
 8:     Dim a As New Ektron.Cms.CommonApi
 9:
10:     'Initalize the IA XML document.
11:     DocumentNode = InformationArchitectureXML.CreateElement("ektron")
12:     InformationArchitectureXML.AppendChild(DocumentNode)
13:     CurrentNode = InformationArchitectureXML.DocumentElement
14:
15:     'Get the project management folder's ID.
16:     PMfolderID = Initialize()
17:
18:     'Get all the child folders of the project management folder (these are "clients").
19:     Try
20:         FolderData = ContentAPI.GetChildFoldersByFolderId(PMfolderID)
21:     Catch ex As Exception
22:         HttpContext.Current.Response.Redirect("Login.aspx", False)
23:         GetInformationArchitectureXML = Nothing
24:         FolderData = Nothing
25:         Exit Function
26:     End Try
27:
28:     If FolderData IsNot Nothing Then
29:         Try
30:             For Each Folder As FolderData In FolderData
31:                 If ContentAPI.MemberType = 1 Then
32:                     If MembershipUserTraverseOnly(ContentAPI.UserId, Folder.Id) = False
Then
33:                         GetProjectXML(CurrentNode, Folder, InformationArchitectureXML)
34:                     End If
35:                 Else
36:                     GetProjectXML(CurrentNode, Folder, InformationArchitectureXML)
37:                 End If
38:             Next
39:         Catch ex As Exception
40:             ErrorNode = InformationArchitectureXML.CreateElement("li")
41:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, ErrorNode, "status",
"error")
42:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, ErrorNode, "message",
"Could not generate information architecture.")
43:             CurrentNode.AppendChild(ErrorNode)
44:         End Try
45:     End If
46:
47:     If IncludeMembers = True Then
```

```
48:         Me.InformationArchitectureXML = InformationArchitectureXML
49:         GetMembersXML()
50:     End If
51:
52:     Return InformationArchitectureXML
53: End Function
```

ProjectManagement.GetIssuesSmartFormID Method

This method returns the ID of the Milestones SmartForm.

C#

```
public int GetIssuesSmartFormID();
```

Visual Basic

```
Public Function GetIssuesSmartFormID() As Integer
```

Returns

Integer

Remarks

The Issue Tracker SmartForm is loaded by RegisterMilestonesSmartForm (see page 87)() from an external XML file.

Body Source

```
 1: Public Function GetIssuesSmartFormID() As Integer
 2:     Dim ContentAPI As New Ektron.Cms.ContentAPI()
 3:     Dim TempID As Integer
 4:     Dim SmartForms As Collection
 5:     Dim SmartForm As Collection
 6:     Dim SmartFormID As Integer = -1
 7:
 8:     'Store the UserID of the actual user in a Temp var.
 9:     TempID = ContentAPI.RequestInformationRef.CallerId
10:     'Set the UserID internal admin.
11:     ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
12:     'Get All SmartForms.
13:     SmartForms = ContentAPI.EkContentRef.GetAllXmlConfigurations("title")
14:     Try
15:         For Each SmartForm In SmartForms
16:             If (SmartForm("CollectionTitle") = "Starterapps.pm.Issues") Then
17:                 SmartFormID = SmartForm("CollectionID")
18:                 Exit Try
19:             End If
20:         Next
21:     Catch ex As Exception
22:         ' handle exception however you want to.
23:     End Try
24:     'Reset users ID.
25:     ContentAPI.RequestInformationRef.CallerId = TempID
26:     Return SmartFormID
27: End Function
```

ProjectManagement.GetMembersXML Method

This method inserts member-information to the Client nodes in the InformationArchitecture.xml generated by GetInformationArchitectureXML (see page 76)().

C#

```
private System.Xml.XmlDocument GetMembersXML();
```

Visual Basic

```
Private Function GetMembersXML() As System.Xml.XmlDocument
```

Returns

InformationArchitecture.xml including membership users assigned to each client.

Remarks

This method inserts a "Members" node to each client. Then, loops through the membership group assigned to the client and inserts a "Member" node (with ID, username, firstname, lastname attributes) for each client to the "Members" node.

Body Source

```
 1: Private Function GetMembersXML() As System.Xml.XmlDocument
 2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
 3:     Dim Clients As System.Xml.XmlNodeList = Nothing
 4:     Dim Users As New Ektron.Cms.API.User.User
 5:     Dim UserGroupData As New Ektron.Cms.UserGroupData
 6:     Dim Members As New Collection
 7:     Dim MembersNode, MemberNode As System.Xml.XmlNode
 8:     Dim XMLUtils As New XMLUtilities
 9:     Dim MemberData As New UserData
10:     Dim TempID As Integer
11:
12:     InformationArchitectureXML = Me.InformationArchitectureXML
13:     If (InformationArchitectureXML IsNot Nothing) Then
14:         Clients = InformationArchitectureXML.SelectNodes("//li[@class='Client']")
15:     End If
16:     If (Clients Is Nothing) Then
17:         Return InformationArchitectureXML
18:         Exit Function
19:     End If
20:     For Each Client As XmlNode In Clients
21:         Try
22:             UserGroupData = Users.GetUserGroupByName("Starterapps.pm." &
Client.Attributes.GetNamedItem("label").Value)
23:         Catch ex As Exception
24:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, Client, "error",
Client.Attributes.GetNamedItem("label").Value)
25:         End Try
26:
27:         If UserGroupData IsNot Nothing Then
28:             'Create a "members" node for this client.
29:             MembersNode = InformationArchitectureXML.CreateElement("li")
30:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MembersNode, "label",
Client.Attributes.GetNamedItem("label").Value & " Members")
31:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MembersNode, "class",
"members")
32:             Client.InsertBefore(MembersNode, Client.FirstChild)
33:
34:             'Store the UserID of the actual user in a Temp var.
35:             TempID = Users.RequestInformationRef.CallerId
36:             'Set the UserID internal admin
37:             Users.RequestInformationRef.CallerId =
Ektron.Cms.Common.EkConstants.InternalAdmin
38:             'Get all users that belong to the client's membership group using Internal
Admin.
39:             Members = Users.EkUserRef.GetUsersByGroupv2_0(UserGroupData.GroupId, "")
40:
41:             If Members IsNot Nothing Then
42:                 For Each Member As Collection In Members
43:                     MemberData = Users.GetUser(Member.Item("UserID"))
44:                     MemberNode = IADoc.CreateElement("li")
45:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"class", "member")
46:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"id", Member.Item("UserID"))
47:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
```

```
"username", System.Web.HttpUtility.HtmlDecode(MemberData.Username))
48:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"firstname", System.Web.HttpUtility.HtmlDecode(MemberData.FirstName))
49:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"lastname", System.Web.HttpUtility.HtmlDecode(MemberData.LastName))
50:                     MembersNode.AppendChild(MemberNode)
51:                 Next
52:             End If
53:
54:             'Reset the UserID to the ID of the actual user.
55:             Users.RequestInformationRef.CallerId = TempID
56:         End If
57:
58:     Next
59:
60:     Return InformationArchitectureXML
61: End Function
```

ProjectManagement.GetMetadataDefinitionID Method

This method gets IDs for any of the three metadata definitions for the Project Management StarterApp.

C#

```
public int GetMetadataDefinitionID(MetadataDefinitionTypes Type);
```

Visual Basic

```
Public Function GetMetadataDefinitionID(ByVal Type As MetadataDefinitionTypes) As Integer
```

Parameters

Parameters	Description
Type	"Type" enumerated to restrict the lookup to the three required metadata definitions required for this application.

Returns

A single metadata definition ID.

Remarks

This method is intended for use with the Project Management StarterApp. This method is not intended to be a general purpose lookup for metadata definition IDs.

Body Source

```
 1: Public Function GetMetadataDefinitionID(ByVal Type As MetadataDefinitionTypes) As Integer
 2:     Dim Metadata As New Metadata
 3:     Dim MetadataCollection() As Ektron.Cms.ContentMetaData
 4:     Dim MetadataType As New Ektron.Cms.ContentMetaData
 5:     Dim SelectedMetadataType As String
 6:
 7:     Select Case Type
 8:         Case MetadataDefinitionTypes.wiki
 9:             SelectedMetadataType = "starterapps.pm.defaultcontent"
10:         Case MetadataDefinitionTypes.cmsblog
11:             SelectedMetadataType = "starterapps.pm.cmsblog"
12:         Case MetadataDefinitionTypes.membershipblog
13:             SelectedMetadataType = "starterapps.pm.membershipblog"
14:         Case Else
15:             SelectedMetadataType = "None"
16:     End Select
17:
18:     MetadataCollection = Metadata.GetMetaDataTypes("Title")
19:     For Each MetadataType In MetadataCollection
20:         If MetadataType.TypeName.ToLower() = SelectedMetadataType.ToLower() Then
21:             GetMetadataDefinitionID = MetadataType.TypeId
22:             Exit For
23:         Else
```

```
24:             GetMetadataDefinitionID = -1
25:         End If
26:     Next
27:
28:     Return GetMetadataDefinitionID
29: End Function
```

ProjectManagement.GetProjectXML Method

This method collects blog component information and inserts this information into InformationArchitecture.xml. This method recursively calls itself to gather all folders/sub-folders of a particular project.

C#

```
private GetProjectXML(XmlNode CurrentNode, Ektron.Cms.FolderData FolderData,
System.Xml.XmlDocument IAdoc);
```

Visual Basic

```
Private Sub GetProjectXML(ByRef CurrentNode As XmlNode, ByRef FolderData As
Ektron.Cms.FolderData, ByRef IAdoc As System.Xml.XmlDocument)
```

Parameters

Parameters	Description
CurrentNode	The current folder in the recursive loop.
FolderData	The Ektron FolderData object of the current folder in the recursive loop.
IAdoc	The pointer to the InformationAcrichtecture.xml document being built.

Remarks

This method fills out the details of project-level components. It collects as XML attributes the client, project or component's...

- **label** - the name of the client, project or component.
- **id** - the folder ID.
- **class** - enumerated as "Client," "Project," "Blog," "DiscussionBoard," "DiscussionForum," "CommunityDocuments, "CommunityMilestones," "Content," "Domain," "Root," or "Community."
- **type** - if the node is a blog, then it also gets a "type" attribute; either "MembershipBlog", or "CMSBlog."

Body Source

```
 1: Private Sub GetProjectXML(ByRef CurrentNode As XmlNode, ByRef FolderData As
Ektron.Cms.FolderData, ByRef IAdoc As System.Xml.XmlDocument)
 2:     Dim Node, ErrorNode As XmlNode
 3:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 4:     Dim Permissions As New Ektron.Cms.API.Permissions
 5:     Dim XMLUtils As New XMLUtilities
 6:     Dim NodeDepth As Integer
 7:     Dim NodeDepthNavigator As System.Xml.XPath.XPathNavigator
 8:     Dim NodeClass As String = "Undetermined"
 9:     Dim GetBlogType As Boolean = False
10:     Dim BlogType As String
11:
12:     'Create a new li element to hold information regarding the current level in the Info
Architecture.
13:     Node = IAdoc.CreateElement("li")
14:
15:     'Add Title and ID attributes.
16:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "label", FolderData.Name)
17:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "id", FolderData.Id)
18:
19:     'Class the node by virtue of its depth in the hierarchy.
20:     NodeDepthNavigator = CurrentNode.CreateNavigator
21:     NodeDepth = NodeDepthNavigator.Evaluate("count(ancestor-or-self::li)")
22:     Select Case NodeDepth
23:         Case 0
```

```
24:             NodeClass = "Client"
25:         Case 1
26:             NodeClass = "Project"
27:         Case Else
28:             Select Case FolderData.FolderType
29:                 Case 0
30:                     NodeClass = "Content"
31:                 Case 1
32:                     NodeClass = "Blog"
33:                     GetBlogType = True
34:                 Case 2
35:                     NodeClass = "Domain"
36:                 Case 3
37:                     NodeClass = "DiscussionBoard"
38:                 Case 4
39:                     NodeClass = "DiscussionForum"
40:                 Case 5
41:                     NodeClass = "Root"
42:                 Case 6
43:                     Select Case FolderData.Name
44:                         Case "Documents"
45:                             NodeClass = "CommunityDocuments"
46:                         Case "Milestones"
47:                             NodeClass = "CommunityMilestones"
48:                         Case "Issues"
49:                             NodeClass = "CommunityIssues"
50:                         Case Else
51:                             NodeClass = "Community"
52:                     End Select
53:             End Select
54:     End Select
55:
56:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "class", NodeClass)
57:
58:     If GetBlogType = True Then
59:         BlogType = GetBlogPermissionsType(FolderData.Id)
60:         XMLUtils.ApplyXMLAttribute(IAdoc, Node, "type", BlogType)
61:     End If
62:
63:     'Append current folder node (with all its attributes, and child node with their
attributes) to its parent node).
64:     CurrentNode.AppendChild(Node)
65:
66:     'Get all child folders that current user has permissions to view.
67:     Dim ChildFolders As New Ektron.Cms.ContentAPI
68:     Dim ChildFolderData() As Ektron.Cms.FolderData = Nothing
69:
70:     Try
71:         ChildFolderData = ChildFolders.GetChildFoldersByFolderId(FolderData.Id)
72:         If ChildFolderData IsNot Nothing Then
73:             For Each ChildFolder As FolderData In ChildFolderData
74:                 If ContentAPI.MemberType = 1 Then
75:                     If MembershipUserTraverseOnly(ContentAPI.UserId, ChildFolder.Id) =
False Then
76:                         GetProjectXML(Node, ChildFolder, IAdoc)
77:                     End If
78:                 Else
79:                     GetProjectXML(Node, ChildFolder, IAdoc)
80:                 End If
81:             Next
82:         End If
83:     Catch ex As Exception
84:         ErrorNode = IAdoc.CreateElement("li")
85:         XMLUtils.ApplyXMLAttribute(IAdoc, ErrorNode, "status", "error")
86:         XMLUtils.ApplyXMLAttribute(IAdoc, ErrorNode, "message", ex.Message)
87:         CurrentNode.AppendChild(ErrorNode)
```

```
88:     End Try
89: End Sub
```

ProjectManagement.GetSmartFormID Method

This method returns the ID of the Milestones SmartForm.

C#

```
public int GetSmartFormID();
```

Visual Basic

```
Public Function GetSmartFormID() As Integer
```

Returns

Integer

Remarks

The Milestones SmartForm is loaded by RegisterMilestonesSmartForm (see page 87)() from an external XML file.

Body Source

```
 1: Public Function GetSmartFormID() As Integer
 2:     Dim ContentAPI As New Ektron.Cms.ContentAPI()
 3:     Dim TempID As Integer
 4:     Dim SmartForms As Collection
 5:     Dim SmartForm As Collection
 6:     Dim SmartFormID As Integer = -1
 7:
 8:     'Store the UserID of the actual user in a Temp var.
 9:     TempID = ContentAPI.RequestInformationRef.CallerId
10:     'Set the UserID internal admin.
11:     ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
12:     'Get All SmartForms.
13:     SmartForms = ContentAPI.EkContentRef.GetAllXmlConfigurations("title")
14:     Try
15:         For Each SmartForm In SmartForms
16:             If (SmartForm("CollectionTitle") = "Starterapps.pm.Milestones") Then
17:                 SmartFormID = SmartForm("CollectionID")
18:                 Exit Try
19:             End If
20:         Next
21:     Catch ex As Exception
22:         ' handle exception however you want to.
23:     End Try
24:     'Reset users ID.
25:     ContentAPI.RequestInformationRef.CallerId = TempID
26:     Return SmartFormID
27: End Function
```

ProjectManagement.Initialize Method

The Initialize() method is executed upon each page load to ensure StartApp has the components it needs, and returns the "Project Management" folder ID.

C#

```
public int Initialize();
```

Visual Basic

```
Public Function Initialize() As Integer
```

Returns

Project Management folder ID.

Remarks

This method...

1. checks to see if the "StarterApps (see page 1)" and "Project Management" folders exists, and if not, creates them.
2. checks to see if the "Starterapps.pm" CMS (see page 1) user group exists, and if not, creates it.
3. checks to see if the three required metadata values "starterapps.pm.DefaultContent", "starterapps.pm.Blog", and "starterapps.pm.PrivateBlog" exist and if not, creates them.
4. checks to see if the required .aspx templates are registered with CMS400 and if not, registers them.

Body Source

```
 1: Public Function Initialize() As Integer
 2:     Dim FolderData() As FolderData
 3:     Dim StarterAppsFolderID As Integer = -1
 4:     Dim PMFolderID As Integer = -1
 5:     Dim CMSGroupID As Integer = -1
 6:     Dim StarterAppsFolderExists As Boolean = False
 7:     Dim PMFolderExists As Boolean = False
 8:     Dim GroupName As String = ""
 9:     Dim PMUser As New StarterApps.ProjectManagement.Users
10:     Dim Metadata As New Metadata
11:     Dim MetadataCollection() As Ektron.Cms.ContentMetaData
12:     Dim MetadataType As New Ektron.Cms.ContentMetaData
13:     Dim MetadataID, SmartFormID As Integer
14:     Dim UserGroup As New API.User.User
15:     Dim MembershipBlogMetadataExists, CMSBlogMetadataExists, WikiMetadataExists As Boolean
16:
17:     'Get all child folders of the CMS Root folder and see if the
18:     'folder "Starter Apps" has already been created.
19:     'If so, set StarterAppsFolderExists to TRUE
20:     FolderData = FolderAPI.GetChildFolders(0, False)
21:     If Not FolderData Is Nothing Then
22:         For Each FolderDataItem As FolderData In FolderData
23:             If FolderDataItem.Name = "Starter Apps" Then
24:                 StarterAppsFolderID = FolderDataItem.Id
25:                 StarterAppsFolderExists = True
26:                 Exit For
27:             End If
28:         Next
29:     End If
30:
31:     Try
32:         'If the "Application Accelerators" folder doesn't exist, create it
33:         If StarterAppsFolderExists = False Then
34:             StarterAppsFolderID = AddFolder("Starter Apps", 0)
35:             'Break inheritance and remove inherited permissions
36:             SetFolderPermissions(StarterAppsFolderID)
37:             'Remove permissions for all inherited groups
38:             RemoveInheritedUserGroups(StarterAppsFolderID)
39:             'Give everyone traverse-only permission on this folder
40:             PMUser.AddUserGroupToFolder(StarterAppsFolderID, EVERYONEGROUP, True)
41:             'Add CMS project management user group and add to "Starter Apps" folder
42:             GroupName = "starterapps.pm"
43:             CMSGroupID = PMUser.AddCMSUserGroup(GroupName)
44:             PMUser.AddUserGroupToFolder(StarterAppsFolderID, CMSGroupID, True)
45:             'Make Folder is marked 'Private'
46:             SetFolderPrivate(StarterAppsFolderID)
47:         End If
48:     Catch ex As Exception
49:         'clean up
50:         StarterAppsFolderID = FolderAPI.GetFolderId("Starter Apps", 0)
51:         If (StarterAppsFolderID <> -1) Then
52:             FolderAPI.DeleteFolderById(StarterAppsFolderID)
```

```
53:         End If
54:         If (CMSGroupID <> -1) Then
55:             UserGroup.DeleteUserGroup(CMSGroupID)
56:         End If
57:         Throw ex
58:         Exit Function
59:     End Try
60:     If (StarterAppsFolderID > 0) Then
61:         'Get all child folders of the "Application Accelerators" folder
62:         'to see if the "Project Management" foldler exists.
63:         FolderData = FolderAPI.GetChildFolders(StarterAppsFolderID, False)
64:         If Not FolderData Is Nothing Then
65:             For Each FolderDataItem As FolderData In FolderData
66:                 If FolderDataItem.Name = "Project Management" Then
67:                     PMFolderID = FolderDataItem.Id
68:                     PMFolderExists = True
69:                     Exit For
70:                 End If
71:             Next
72:         End If
73:
74:
75:         'If the PM folder doesn't exist, folder doesn't exist, create it.
76:         If PMFolderExists = False Then
77:             PMFolderID = AddFolder("Project Management", StarterAppsFolderID)
78:             'Break inheritance and remove inherited permissions.
79:             SetFolderPermissions(PMFolderID)
80:             'Remove permissions for all inherited groups.
81:             RemoveInheritedUserGroups(PMFolderID)
82:             'Add CMS user group and add to PM folder.
83:             GroupName = "starterapps.pm"
84:             CMSGroupID = PMUser.AddCMSUserGroup(GroupName)
85:             PMUser.AddUserGroupToFolder(PMFolderID, CMSGroupID, False)
86:             'Make Folder is marked 'Private'.
87:             SetFolderPrivate(PMFolderID)
88:         End If
89:
90:         'See if the metadata mefinition for default content exists.
91:         'If not, create it.
92:
93:         MembershipBlogMetadataExists = False
94:         CMSBlogMetadataExists = False
95:         WikiMetadataExists = False
96:
97:         MetadataCollection = Metadata.GetMetaDataTypes("Title")
98:         For Each MetadataType In MetadataCollection
99:             Select Case MetadataType.TypeName.ToLower()
100:                Case "starterapps.pm.defaultcontent"
101:                    MetadataID = MetadataType.TypeId
102:                    WikiMetadataExists = True
103:                Case "starterapps.pm.membershipblog"
104:                    MetadataID = MetadataType.TypeId
105:                    MembershipBlogMetadataExists = True
106:                Case "starterapps.pm.cmsblog"
107:                    MetadataID = MetadataType.TypeId
108:                    CMSBlogMetadataExists = True
109:            End Select
110:        Next
111:
112:        If WikiMetadataExists = False Then
113:            MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.Wiki)
114:        End If
115:
116:        If MembershipBlogMetadataExists = False Then
117:            MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.MembershipBlog)
118:        End If
```

```
119:
120:         If CMSBlogMetadataExists = False Then
121:             MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.CMSBlog)
122:         End If
123:
124:         'Register Page Templates.
125:         RegisterTemplate("Projects.aspx")
126:         RegisterTemplate("Wiki.aspx")
127:         RegisterTemplate("Dynamic.aspx")
128:         RegisterTemplate("Discussion.aspx")
129:         RegisterTemplate("Documents.aspx")
130:         RegisterTemplate("Milestones.aspx")
131:         RegisterTemplate("Blog.aspx")
132:         RegisterTemplate("Login.aspx")
133:         RegisterTemplate("Issues.aspx")
134:
135:         'Import Milestones Smartform.
136:         SmartFormID = GetSmartFormID()
137:         If SmartFormID = -1 Then
138:             RegisterMilestonesSmartForm()
139:         End If
140:
141:         'Import Issues Smartform.
142:         SmartFormID = GetIssuesSmartFormID()
143:         If SmartFormID = -1 Then
144:             RegisterIssuesSmartForm()
145:         End If
146:     End If
147:     Return PMFolderID
148: End Function
```

ProjectManagement.MembershipUserTraverseOnly Method

This method checks to see if a Membership user ID has been specifically granted permissions to a folder.

C#

```
private Boolean MembershipUserTraverseOnly(int UserId, int FolderID);
```

Visual Basic

```
Private Function MembershipUserTraverseOnly(ByVal UserId As Integer, ByVal FolderID As
Integer) As Boolean
```

Parameters

Parameters	Description
UserId	The Membership user's ID.
FolderID	The folder ID of the folder we wish to check.

Returns

True - if the user can only Traverse the folder.

False - if the user has been granted permissions to interact (e.g. add, delete, edit...) with the content in the folder.

Remarks

By default, Membership users have Traverse permissions for all folders. This is a hardcoded setting! This method figures out if a Membership user has been specifically granted permission for a folder via direct assignment of the user to the folder, or via a Membership user group that has been assigned to the folder. If the user has permissions beyond Traverse, the return value is set to "False", else it is set to "True."

Body Source

```
 1: Private Function MembershipUserTraverseOnly(ByVal UserId As Integer, ByVal FolderID As
Integer) As Boolean
 2:     Dim UserAPI As New Ektron.Cms.API.User.User
```

```
 3:     Dim GroupData() As Ektron.Cms.GroupData
 4:     Dim PermissionsAPI As New Ektron.Cms.API.Permissions
 5:     Dim PermissionsData() As Ektron.Cms.UserPermissionData
 6:
 7:     'Get all the groups to which the user is assigned.
 8:     GroupData = UserAPI.GetGroupsUserIsIn(UserId, "groupname")
 9:     'Get the permissions data for the folder being examined.
10:     PermissionsData = PermissionsAPI.GetUserPermissions(FolderID, "folder", 0, "")
11:     'Set the default return value - that the Membership user has ONLY traverse permissions.
12:     MembershipUserTraverseOnly = True
13:
14:     For Each PermissionsItem As UserPermissionData In PermissionsData
15:         If PermissionsItem IsNot Nothing Then
16:             'Check to see if the user has been granted access to the folder directly.
17:             If PermissionsItem.UserId = UserId Then
18:                 'In this case, the user has specifically been granted access to the folder.
19:                 'Set return value to false - user has been granted more access
20:                 'than the default traverse-only permission.
21:                 MembershipUserTraverseOnly = False
22:                 Exit For
23:             End If
24:             'Check to see if the user has been granted access to the folder via a
Membership group.
25:             For Each UserGroup As GroupData In GroupData
26:                 If UserGroup.GroupId = PermissionsItem.GroupId Then
27:                     'In this case, the user belongs to a Membership user group that has
28:                     'specifically been granted access to the folder.
29:                     'Set return value to false - user has been granted more access
30:                     'than the default traverse-only permission.
31:                     MembershipUserTraverseOnly = False
32:                     Exit For
33:                 End If
34:             Next
35:         End If
36:         If MembershipUserTraverseOnly = False Then
37:             Exit For
38:         End If
39:     Next
40: End Function
```

ProjectManagement.RegisterMilestonesSmartForm Method

Imports the Milestones SmartFrom from "xml/MilestonesSmartForm.xml"

C#

```
public RegisterMilestonesSmartForm();
```

Visual Basic

```
Public Sub RegisterMilestonesSmartForm()
```

Remarks

You can later get the ID of this SmartForm via GetSmartFormID (see page 83)().

Body Source

```
 1: Public Sub RegisterMilestonesSmartForm()
 2:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 3:     Dim TempID As Integer
 4:     Dim SmartFormData As New Collection
 5:     Dim SmartFormXML, SmartFormDisplayXSLT As String
 6:     Dim XMLDoc, XSLDoc As New System.Xml.XmlDocument
 7:     Dim SmartFormID As Integer
 8:     Dim EkContent As New Ektron.Cms.Content.EkContent
 9:     Dim Package As New Collection()
10:
```

```
11:     'Store the current users's ID in a temprorary integer var.
12:     TempID = ContentAPI.RequestInformationRef.CallerId
13:     'Set the current user's ID to the INTERNALADMIN constant.
14:     ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
15:
16:     'Get Milestones SmartForm Package XML.
17:
XMLDoc.Load(System.Web.HttpContext.Current.Server.MapPath("xml/MilestonesSmartForm.xml"))
18:     SmartFormXML = XMLDoc.InnerXml
19:     'Get Milestones SmartForm Display XSLT.
20:
XSLDoc.Load(System.Web.HttpContext.Current.Server.MapPath("xml/MilestonesSmartForm.xsl"))
21:     SmartFormDisplayXSLT = XSLDoc.InnerXml
22:
23:     'Below are the Collection Paramters for the SmartForm Collection.
24:     SmartFormData.Add("0", "DefaultXSLT")
25:     SmartFormData.Add("Starterapps.pm.Milestones", "CollectionTitle")
26:     SmartFormData.Add("This smart form is associated with Milestones in the Project
Management StarterApp", "CollectionDescription")
27:     SmartFormData.Add(System.Web.HttpContext.Current.Server.MapPath("xml"), "PhysicalPath")
28:     SmartFormData.Add("", "displayXslt")
29:     SmartFormData.Add("", "EditXSLT")
30:     SmartFormData.Add("", "SaveXSLT")
31:     SmartFormData.Add("", "XSLT1")
32:     SmartFormData.Add("", "XSLT2")
33:     SmartFormData.Add("", "XSLT3")
34:     SmartFormData.Add("", "XSLT4")
35:     SmartFormData.Add("", "XSLT5")
36:     SmartFormData.Add("", "XmlSchema")
37:     SmartFormData.Add("", "XmlNameSpace")
38:     SmartFormData.Add("", "XmlAdvConfig")
39:     SmartFormID = ContentAPI.AddXmlConfiguration(SmartFormData)
40:
41:     'Add the SmartForm Package.
42:     Package.Add(SmartFormID, "XmlCollectionID")
43:     Package.Add(SmartFormXML, "Package")
44:     Package.Add(SmartFormDisplayXSLT, "displayXslt")
45:     Package.Add("", "DesignStylesheet")
46:     ContentAPI.UpdatexmlCollectionPackage(Package)
47:
48:     'Set the current user's ID back.
49:     ContentAPI.RequestInformationRef.CallerId = TempID
50: End Sub
```

ProjectManagement.RegisterTemplate Method

This method registers our templates inside CMS400 during Project Management StarterApp Initialization.

C#

```
public RegisterTemplate(String TemplateName);
```

Visual Basic

```
Public Sub RegisterTemplate(ByVal TemplateName As String)
```

Parameters

Parameters	Description
TemplateName	The name of the template to resister.

Remarks

Path to the templates is hard coded in this method as "StarterApps (see page 1)/ProjectManagement (see page 71)/"

Body Source

```
1: Public Sub RegisterTemplate(ByVal TemplateName As String)
2:     Dim TemplateData As New Collection()
```

```
    Dim ContentAPI As New Ektron.Cms.ContentAPI
    Dim ExistingTemplates As TemplateData() = ContentAPI.GetAllTemplates("")
    Dim TemplateFileName As String
    Dim TemplateExists As Boolean = False

    TemplateFileName = "StarterApps/ProjectManagement/" & TemplateName

    'Check to see if template already exists
    For Each Template As TemplateData In ExistingTemplates
        If Template.FileName = TemplateFileName Then
            TemplateExists = True
            Exit For
        End If
    Next

    'If the template doesn't exist, add it
    If TemplateExists = False Then
        TemplateData.Add(TemplateFileName, "TemplateFileName")
        ContentAPI.EkContentRef.AddTemplatev2_0(TemplateData)
    End If
End Sub
```

ProjectManagement.RemoveInheritedUserGroups Method

This method removes any user groups and users assigned to a folder.

C#

```
public RemoveInheritedUserGroups(int FolderID);
```

Visual Basic

```
Public Sub RemoveInheritedUserGroups(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	The ID of the folder from which to remove all user groups.

Remarks

This method is used to "clean up" a folder after breaking content inheritance and before adding new user groups to create a new permissions model.

Notes

This method removes both all users and all user groups.

Body Source

```
Public Sub RemoveInheritedUserGroups(ByVal FolderID As Integer)
    Dim Permissions As New Ektron.Cms.API.Permissions
    Dim PermissionsData() As Ektron.Cms.UserPermissionData

    PermissionsData = Permissions.GetUserPermissions(FolderID, "folder", 0, "")

    If PermissionsData IsNot Nothing Then
        For Each UserGroup As UserPermissionData In PermissionsData
            Select Case UserGroup.GroupId
                Case -1
                    If UserGroup.UserId > 1 Then
                        'This is a user - delete it.
                        Permissions.DeleteItemPermission(UserGroup.UserId, False,
FolderID, EkEnumeration.CMSObjectTypes.Folder)
                    End If
                Case 1
                    'This is the admin group - do nothing.
                Case Else
                    'This is a group - delete it.
```

```
19:                     Permissions.DeleteItemPermission(UserGroup.GroupId, True, FolderID,
EkEnumeration.CMSObjectTypes.Folder)
20:             End Select
21:         Next
22:     End If
23: End Sub
```

ProjectManagement.SetFolderPermissions Method

This method breaks content and metadata inheritance for folders. You may specify which inheritance property you wish to break or specify if you wish to break both.

C#

```
public SetFolderPermissions(int FolderID);
```

Visual Basic

```
Public Sub SetFolderPermissions(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	Folder ID of the folder you wish to break inheritance.

Remarks

Breaking folder inheritance allows us to apply the three required Project Management StarterApp metadata types only to the folders that require them. Only Blogs should be assigned the "blogs" metadata types. The "project/wiki (see page 130)" folder should only be assigned the "default content" metadata type

Body Source

```
1: Public Sub SetFolderPermissions(ByVal FolderID As Integer)
2:     Dim PermissionData As New Ektron.Cms.UserPermissionData
3:     Dim Permissions As New Ektron.Cms.API.Permissions
4:
5:     'Break Folder Inheritance
6:     Permissions.DisableFolderInheritance(FolderID)
7: End Sub
```

ProjectManagement.SetFolderPrivate Method

This method sets the folder to private status.

C#

```
public SetFolderPrivate(int FolderID);
```

Visual Basic

```
Public Sub SetFolderPrivate(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	Folder ID of the folder to make private.

Remarks

This is a workaround since there is no direct API call to ensure a folder is set to private. To ensure this method executes properly, we must borrow the internal admin's rights by switching from the user's ID who is setting the folder to private to the constant INTERNALADMIN. Then, switch it back before finishing execution. Calling this method without setting the internal admin, even with an authorized CMS (see page 1) user, results in throwing an error. Since this user has already been authenticated to perform this action (creating a client or project folder), we can use internal admin to avoid this problem.

Body Source

```
1: Public Sub SetFolderPrivate(ByVal FolderID As Integer)
```

```
    Dim ContentAPI As New Ektron.Cms.ContentAPI
    Dim EkContent As Ektron.Cms.Content.EkContent = ContentAPI.EkContentRef
    Dim PageData As New Collection
    Dim TempID As Integer

    'Store the current users's ID in a temprorary integer var.
    TempID = ContentAPI.RequestInformationRef.CallerId
    'Set the current user's ID to the INTERNALADMIN constant.
    ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
    PageData.Add(FolderID, "ItemID")
    PageData.Add("folder", "RequestType")

    'EkContent.DisableItemPrivateSettingv2_0(pagedata) ' to make public
    EkContent.EnableItemPrivateSettingv2_0(PageData) ' to make private

    'Set the current user's ID back
    ContentAPI.RequestInformationRef.CallerId = TempID
End Sub
```

ProjectManagement.TransformInformationArchitectureXML Method

This method transforms Information Architecture XML to an XHTML unordered list.

C#

```
public String TransformInformationArchitectureXML();
```

Visual Basic

```
Public Function TransformInformationArchitectureXML() As String
```

Returns

An XHTML unordered list.

Remarks

This method is used to create the client/project navigation tree that the current user has permissions to traverse.

Body Source

```
Public Function TransformInformationArchitectureXML() As String
    Dim XMLUtils As New XMLUtilities

    'Set XML Args.
    XMLUtils.XMLSourceType = XMLUtilities.XMLSourceTypes.XMLDocument
    XMLUtils.XMLDocument = InformationArchitectureXML
    'Set XSLT Params.
    XMLUtils.XSLTParams.Add("UserMode", Me.InformationArchitectureUserMode)
    XMLUtils.XSLTParams.Add("ClientID", Me.ClientID)
    XMLUtils.XSLTParams.Add("ProjectID", Me.ProjectID)
    'Set XSLT Args.
    XMLUtils.XSLTSourceType = XMLUtilities.XSLTSourceTypes.File
    XMLUtils.XSLTSource = Me.IAPath & "xml/InformationArchitecture.xslt"
    'Transform Information Architectre.
    TransformInformationArchitectureXML = XMLUtils.TransformXML()
End Function
```

ProjectManagement Properties

The properties of the ProjectManagement class are listed here.

Public Properties

	Name	Description
	ClientID (see page 92)	The ClientID property is the FolderID of the selected client and is consumed by several methods in the ProjectManagement (see page 71) namespace.
	IAPath (see page 92)	The IAPath Property indicates the location of InformationArchitecture.xslt.

	InformationArchitectureUserMode (see page 92)	This Property is used to determine "action" rights for users and is consumed by InformationArchitecture.xslt when called by the "TransformInformationArchitetureXML()." Default value is "MembershipUser."
	InformationArchitectureXML (see page 93)	The InformationArchitectureXML property is used to hold the the information architecture prior to being transformed with XSLT.
	ProjectID (see page 93)	The ProjectID property is consumed by several XMLXSLT transformations.
	Template (see page 93)	The Template property is used in several XSLT transformations to identify the aspx template associated with a link.
	TemplateLabel (see page 94)	The TemplateLabel property indicates the text to display for an aspx template associated with a particular link. This property is consumed by several XSLT transformations.

Legend

	Property

ProjectManagement.ClientID Property

The ClientID property is the FolderID of the selected client and is consumed by several methods in the ProjectManagement (see page 71) namespace.

C#

```
public int ClientID;
```

Visual Basic

```
Public Property ClientID() As Integer
```

Returns

FolderID of the selected client.

Remarks

This is used all over the place.

ProjectManagement.IAPath Property

The IAPath Property indicates the location of InformationArchitecture.xslt.

C#

```
public String IAPath;
```

Visual Basic

```
Public Property IAPath() As String
```

Returns

Relative location path - e.g. "../"

Description

String

Remarks

The Information Architecture XSLT is called from several physical locations; site root, and from AJAX components under "/actions." This property allows a method calling this XSLT to identify its relative location to InformationArchitecture.xslt.

ProjectManagement.InformationArchitectureUserMode Property

This Property is used to determine "action" rights for users and is consumed by InformationArchitecture.xslt when called by the "TransformInformationArchitetureXML()." Default value is "MembershipUser."

C#

```
public InformationArchitectureUserModes InformationArchitectureUserMode;
```

Visual Basic

```
Public Property InformationArchitectureUserMode() As InformationArchitectureUserModes
```

ProjectManagement.InformationArchitectureXML Property

The InformationArchitectureXML property is used to hold the the information architecture prior to being transformed with XSLT.

C#

```
public System.Xml.XmlDocument InformationArchitectureXML;
```

Visual Basic

```
Public Property InformationArchitectureXML() As System.Xml.XmlDocument
```

Returns

The Information Architecture XmlDocument.

Description

XmlDocument

Remarks

IA XML is transformned in several places - to generate the client list, to get the breadcrumbs, to fill out links in the project navigation widget.

ProjectManagement.ProjectID Property

The ProjectID property is consumed by several XMLXSLT transformations.

C#

```
public int ProjectID;
```

Visual Basic

```
Public Property ProjectID() As Integer
```

Returns

FolderID of the selected project.

Remarks

There is an application-level variable, `Ektron_StarterApps_ProjectManagement_ProjectID_~integer~`, that indicates the current project. This is set in Main.master.vb. This variable ensures that the current project's folder ID is always available; even if it's not part of the querystring.

ProjectManagement.Template Property

The Template property is used in several XSLT transformations to identify the aspx template associated with a link.

C#

```
public String Template;
```

Visual Basic

```
Public Property Template() As String
```

Returns

The ASPX template associated with a link.

Description

String

Remarks

This is the template target of a link. See property TemplateLabel (see page 94)().

ProjectManagement.TemplateLabel Property

The TemplateLabel property indicates the text to display for an aspx template associated with a particular link. This property is consumed by several XSLT transformations.

C#

```
public String TemplateLabel;
```

Visual Basic

```
Public Property TemplateLabel() As String
```

Returns

The display string for a link template.

Description

String

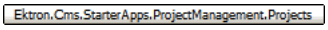
Remarks

This label is what shows up to the user.

Projects Class

Contains methods and properties that manage the project aspect of the project management site. This includes functionality for Blogs, Discussion Boards, Taxonomy, Wiki (see page 130) and Folders.

Class Hierarchy



C#

```
public class Projects;
```

Visual Basic

```
Public Class Projects
```

File

ProjectManagement.vb

Projects Methods

The methods of the Projects class are listed here.

Private Methods

	Name	Description
	DefaultBlogContentText (see page 106)	Returns default Blog content.
	DefaultWikiContentText (see page 106)	Returns default Wiki (see page 130) content.
	GetDefaultTaxonomy (see page 108)	This method loads the default taxonomy ("DefaultProjectTaxonomy.xml") into a string.
	LoremIpsumLong (see page 109)	This method generates placeholder text.
	LoremIpsumShort (see page 110)	This method generates placeholder text.

	MakeFolderNameUnique (see page 110)	This method ensures a project name is unique.
	MakeFolderReadOnly (see page 111)	This method is for use with the Milestones folder in the Project Management Starter App.

Legend

	Method
	Private

Protected Methods

	Name	Description
	AddBlog (see page 95)	This method creates a blog for the Project Management StarterApp.
	AddDefaultBlogEntry (see page 96)	This method creates a default blog entry as a placeholder (Lorem Ipsum text).
	AddDiscussion (see page 97)	This method adds a discussion to the Project Management StarterApp.
	AddFolder (see page 98)	This method adds a CMS (see page 1) folder to the Project Management StarterApp.

Legend

	Method
	Protected

Public Methods

	Name	Description
	AddMetadataDefinitionToFolder (see page 99)	This method assigns a metadata definition (by ID) to a folder (by ID).
	AddPMCommunityFolder (see page 99)	This method adds a Community folder to the Project Management StarterApp.
	AddProject (see page 100)	This method adds a project to the Project Management StarterApp.
	GetAllProjects (see page 107)	This method retrieves all project folder names and IDs that the user has permissions to view.
	GetProjectTaxonomyID (see page 108)	Gets the ID of the taxonomy associated with the project folder.
	RemoveProject (see page 112)	This method removes a project from the Project Management StarterAPP.
	SetFolderTemplate (see page 113)	This method sets the default .aspx template for a folder.

Legend

	Method

Projects.AddBlog Method

This method creates a blog for the Project Management StarterApp.

C#

```
protected int AddBlog(int ParentFolderID, String BlogName, String BlogTitle, String
BlogDescription);
```

Visual Basic

```
Protected Function AddBlog(ByVal ParentFolderID As Integer, ByVal BlogName As String, ByVal
BlogTitle As String, ByVal BlogDescription As String) As Integer
```

Parameters

Parameters	Description
ParentFolderID	The ID of the folder that will become the parent to the blog to be added.
BlogName	The name of the blog.
BlogTitle	The title of the blog.
BlogDescription	The description of the blog. (This can be empty string "")

Returns

The ID of the newly created blog.

Remarks

Some details regarding how blogs are being created in this method...

1. Blog Visibility is set to private.
2. EnableComments is set to True.
3. ModerateComments is set to False.
4. CommentsRequireAuthentication is set to True.
5. No categories are added.
6. No blogroll information is added.

Body Source

```
 1: Protected Function AddBlog(ByVal ParentFolderID As Integer, ByVal BlogName As String,
ByVal BlogTitle As String, ByVal BlogDescription As String) As Integer
 2:     Dim Name, Title, Description As String
 3:     Dim Categories(1) As String
 4:     Dim Blogroll As New Ektron.Cms.BlogRoll 'This is empty.
 5:     Dim EnableComments, ModerateComments, CommentsRequireAuthentication As Boolean
 6:     Dim BlogID As Integer
 7:     Dim NewBlog As New Ektron.Cms.API.Content.Blog
 8:
 9:
10:     Name = BlogName
11:     Title = BlogTitle
12:     Description = BlogDescription
13:     EnableComments = True
14:     ModerateComments = False
15:     CommentsRequireAuthentication = True
16:     Categories(0) = "Status"
17:     Categories(1) = "Questions"
18:
19:     BlogID = NewBlog.AddBlog(ParentFolderID, Name, Title, Description,
EkEnumeration.BlogVisibility.Private, EnableComments, ModerateComments,
CommentsRequireAuthentication, Categories, Blogroll)
20:
21:     Return BlogID
22: End Function
```

Projects.AddDefaultBlogEntry Method

This method creates a default blog entry as a placeholder (Lorem Ipsum text).

C#

```
protected AddDefaultBlogEntry(int BlogID, String BlogName);
```

Visual Basic

```
Protected Sub AddDefaultBlogEntry(ByVal BlogID As Integer, ByVal BlogName As String)
```

Parameters

Parameters	Description
BlogID	The ID of the blog where the post will be added.
BlogName	The name of the blog (this is used in the post's title).

Remarks

This method exists to create placeholder text.

Body Source

```
 1: Protected Sub AddDefaultBlogEntry(ByVal BlogID As Integer, ByVal BlogName As String)
 2:     Dim PostContent As New Ektron.Cms.ContentData
 3:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 4:     Dim Categories(0) As String
 5:     Dim NewBlog As New Ektron.Cms.API.Content.Blog
 6:
 7:     'Add Default Blog Entry
 8:     PostContent.ContType = Ektron.Cms.Common.EkConstants.CMSContentType_Content
 9:     PostContent.DateCreated = Now()
10:     PostContent.Html = DefaultBlogContentText()
11:     PostContent.Title = "Default " & BlogName & " Entry"
12:     PostContent.Teaser = LoremIpsumShort()
13:     PostContent.UserId = ContentAPI.UserId
14:     PostContent.LanguageId = ContentAPI.DefaultContentLanguage
15:     PostContent.IsSearchable = True
16:     NewBlog.AddPost(BlogID, PostContent, Categories, False, "", "")
17: End Sub
```

Projects.AddDiscussion Method

This method adds a discussion to the Project Management StarterApp.

C#

```
protected int AddDiscussion(int ParentFolderID);
```

Visual Basic

```
Protected Function AddDiscussion(ByVal ParentFolderID As Integer) As Integer
```

Parameters

Parameters	Description
ParentFolderID	The ID of the folder that will become the parent to the blog to be added.

Returns

The ID of the newly created discussion.

Remarks

Some details regarding how discussions are being created in this method...

1. Name is always "Discussion."
2. Title is always "~project name~ Discussion."
3. Authentication is set to True.
4. A default category is added ("~project name~ General Discussion").
5. ModeratePosts is set to False.
6. LockForum is set to False.
7. SortOrder is set to 1. This sets the order of forums added. Since there's only one forum, this param doesn't mean much, but it is required.

Body Source

```
 1: Protected Function AddDiscussion(ByVal ParentFolderID As Integer) As Integer
 2:     Dim Name, Title As String
 3:     Dim Authentication As Boolean
 4:     Dim Categories(0) As String
 5:     Dim BoardID As Integer
 6:     Dim BoardCategories() As Ektron.Cms.DiscussionCategory
 7:     Dim BoardCategory As New Ektron.Cms.DiscussionCategory
 8:     Dim i As Integer = 0
 9:
```

```
10:     Name = "Discussion"
11:     Title = Me.ProjectName & " Discussion"
12:     Authentication = True
13:     Categories(0) = Me.ProjectName & " General Discussion"
14:
15:     Try
16:         Dim Discussion As New Ektron.Cms.API.Content.ThreadedDiscussion
17:         BoardID = Discussion.AddBoard(ParentFolderID, Name, Title, Authentication, "",
Categories)
18:         BoardCategories = Discussion.GetBoardCategories(BoardID)
19:
20:         If Not BoardCategories Is Nothing Then
21:             For Each BoardCategory In BoardCategories
22:                 Discussion.AddForum(BoardID, Me.ProjectName & " Forum", Me.ProjectName & "
General Discussion", False, False, 1, BoardCategory.CategoryID)
23:             Next
24:         End If
25:     Catch ex As Exception
26:         BoardID = -1
27:     End Try
28:
29:     Return BoardID
30: End Function
```

Projects.AddFolder Method

This method adds a CMS (see page 1) folder to the Project Management StarterApp.

C#

```
protected int AddFolder(int ParentFolderID, String FolderName, String FolderDescription);
```

Visual Basic

```
Protected Function AddFolder(ByVal ParentFolderID As Integer, ByVal FolderName As String,
ByVal FolderDescription As String) As Integer
```

Parameters

Parameters	Description
ParentFolderID	The ID of the folder that will become the parent to the folder to be added.
FolderName	The name of the folder to be added.
FolderDescription	The description of the folder to be added. (This can be an empty string.)

Returns

Returns the ID of the newly created CMS (see page 1) folder.

Remarks

This is intended to be used only when creating the "StarterApps (see page 1)," "ProjectManagement (see page 71)," and client folders, but will work for any folder ID (provided user has access to folder). All other content folders in Project Management StarterApp are Community folders.

Body Source

```
 1: Protected Function AddFolder(ByVal ParentFolderID As Integer, ByVal FolderName As String,
ByVal FolderDescription As String) As Integer
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim FolderRequest As New FolderRequest
 4:
 5:     'Create Folder
 6:     FolderRequest.FolderName = FolderName
 7:     FolderRequest.ParentId = ParentFolderID
 8:     FolderRequest.FolderDescription = FolderDescription
 9:     FolderRequest.SiteMapPathInherit = False
10:     FolderRequest.MetaInherited = 0
11:     FolderAPI.AddFolder(FolderRequest)
```

```
12:
13:     Return FolderRequest.FolderId
14: End Function
```

Projects.AddMetadataDefinitionToFolder Method

This method assigns a metadata definition (by ID) to a folder (by ID).

C#

```
public AddMetadataDefinitionToFolder(int FolderID, int MetadataDefinitionID);
```

Visual Basic

```
Public Sub AddMetadataDefinitionToFolder(ByVal FolderID As Integer, ByVal MetadataDefinitionID
As Integer)
```

Parameters

Parameters	Description
FolderID	The ID of the folder to assign the metadata definition.
MetadataDefinitionID	The ID of the metadata definition to assign to the folder.

Body Source

```
 1: Public Sub AddMetadataDefinitionToFolder(ByVal FolderID As Integer, ByVal
MetadataDefinitionID As Integer)
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim content As Ektron.Cms.Content.EkContent
 4:     Dim api As New Ektron.Cms.CommonApi
 5:     Dim tmpUserID As Integer = 0
 6:     Dim tmpUserToken As Integer = 0
 7:     tmpUserID = api.RequestInformationRef.CallerId
 8:     tmpUserToken = api.RequestInformationRef.UniqueId
 9:     Try
10:         api.RequestInformationRef.CallerId = EkConstants.InternalAdmin
11:         content = api.EkContentRef
12:         'Assign metadata definition to folder.
13:         content.ProcessCustomFields(FolderID, "false", "Assigned_" & MetadataDefinitionID,
FolderAPI.GetFolder(FolderID).ParentId)
14:     Finally
15:         api.RequestInformationRef.CallerId = tmpUserID
16:         api.RequestInformationRef.UniqueId = tmpUserToken
17:     End Try
18: End Sub
```

Projects.AddPMCommunityFolder Method

This method adds a Community folder to the Project Management StarterApp.

C#

```
public int AddPMCommunityFolder(int ParentFolderID, String FolderName, String
FolderDescription, int MetadataDefinitionID);
```

Visual Basic

```
Public Function AddPMCommunityFolder(ByVal ParentFolderID As Integer, ByVal FolderName As
String, ByVal FolderDescription As String, ByVal MetadataDefinitionID As Integer) As Integer
```

Parameters

Parameters	Description
ParentFolderID	The ID of the folder that will become the parent to the folder to be added.
FolderName	The name of the folder to be added.
FolderDescription	The description of the folder to be added. (This can be an empty string.)
MetadataDefinitionID	The ID of a metadata definition to be assigned to the folder.

Returns

Returns the ID of the newly created CMS (see page 1) folder.

Remarks

The metadata definition can be for general purpose but is intended to be used specifically with Wikis. In the Project Management StarterApp, when a user navigates to the Wiki (see page 130), we need to load a content block by default. The first step in figuring out which content block is the default for the selected Wiki (see page 130) is to assign the "Starterapps.pm.DefaultContent" metadata definition to the Wiki (see page 130) folder. Step two is to assign this metadata to a content block in the Wiki (see page 130). Step three is to load the (hopefully) one content block that has been assigned the "Starterapps.pm.DefaultContent" metadata value. Besides that, this method simply creates a Community Folder.

Body Source

```
1: Public Function AddPMCommunityFolder(ByVal ParentFolderID As Integer, ByVal FolderName As String, ByVal FolderDescription As String, ByVal MetadataDefinitionID As Integer) As Integer
2:     Return AddPMCommunityFolder(ParentFolderID, FolderName, FolderDescription, MetadataDefinitionID, "", -1)
3: End Function
```

Projects.AddProject Method

This method adds a project to the Project Management StarterApp.

C#

```
public AddProject();
```

Visual Basic

```
Public Sub AddProject()
```

Remarks

This method does several things...

1. Creates the Project folder (this is the Wiki (see page 130) folder) as a Community Folder and associates this folder with "Wiki.aspx."
2. Breaks inheritance and sets the Project folder to PRIVATE.
3. Creates a default Wiki (see page 130) project entry (a few paragraphs of Lorem Ipsum).
4. Imports the default PM Taxonomy for use with the Wiki (see page 130).
5. Creates the Documents folder as Community Folder and associates this folder with "Documents.aspx."
6. Breaks inheritance and sets the Documents folder to PRIVATE.
7. Creates the Milestones folder as Community Folder and associates this folder with "Milestones.aspx."
8. Breaks inheritance and sets the Milestones folder to PRIVATE.
9. Creates the MembershipBlog and associates this folder with "Blog.aspx."
10. Breaks inheritance and sets permissions for this folder.
11. Creates the CMSBlog and associates this folder with "Blog.aspx."
12. Breaks inheritance and sets permissions for this folder.
13. Creates the Discussion folder and associates this folder with "Discussion.aspx."
14. Breaks inheritance and sets permissions for this folder.

Body Source

```
1: Public Sub AddProject()
2:
3:     'Ektron.ProjectManagement Namespace
```

```
    Dim PM As New Ektron.Cms.StarterApps.ProjectManagement.ProjectManagement
    Dim Clients As New Clients
    Dim Users As New Users

    'Ektron
    Dim NewProjectFolder As New FolderRequest
    Dim FolderRequest As New Ektron.Cms.FolderRequest
    Dim FolderData As New Ektron.Cms.FolderData
    Dim FolderAPI As New Ektron.Cms.API.Folder
    Dim ClientGroup As New Ektron.Cms.API.User.User
    Dim ContentAPI As New Ektron.Cms.ContentAPI
    Dim Taxonomy As New Ektron.Cms.API.Content.Taxonomy
    Dim TaxonomyBaseDatum As New Ektron.Cms.TaxonomyBaseData
    Dim TaxonomySyncRequest As New Ektron.Cms.TaxonomySyncRequest
    Dim TaxonomyContentRequest As New TaxonomyContentRequest
    Dim TaxonomyRequest As New Ektron.Cms.TaxonomyRequest
    Dim UserGroup As New API.User.User
    Dim UserGroupData As New UserGroupData
    Dim Permissions As New Ektron.Cms.API.Permissions

    'General
    'Dim PMFolderID, ClientFolderID, ProjectFolderID, DocumentsFolderID,
MilestonesFolderID, CMSGroupID, BlogID, DiscussionID, MetadataDefinitionID, TaxonomyID,
MilestonesSmartFormID, DefaultWikiContentID, TaxonomyCategoryID As Integer
    Dim TaxonomyID, TaxonomyCategoryID As Integer
    Dim PMFolderID, ClientFolderID, ProjectFolderID, DocumentsFolderID, IssuesFolderID,
MilestonesFolderID, CMSGroupID, BlogID, DiscussionID, MetadataDefinitionID, IssuesSmartFormID,
MilestonesSmartFormID, DefaultWikiContentID As Integer

    Dim DefaultTaxonomy, TaxonomyPath As String

    Dim SmartFormCollection, BlogPermissionData As New Collection()

    'Get PM folder ID.
    PMFolderID = PM.Initialize
    PM.Initialize()
    'Get Client folder ID.
    ClientFolderID = Me.ClientID
    'Set ClientName property.
    FolderData = FolderAPI.GetFolder(ClientFolderID)
    Me.ClientName = FolderData.Name
    'Get CMS group ID.
    CMSGroupID = Users.AddCMSUserGroup("starterapps.pm")

    '- Project Folder -
    '-------------------------------------------------------------------------------
    'Get metadta definition ID for default content for wiki (so we can assign a "default"
content block).
    MetadataDefinitionID =
PM.GetMetadataDefinitionID(ProjectManagement.MetadataDefinitionTypes.Wiki)

    'Add Project folder.
    Me.ProjectName = MakeFolderNameUnique(ClientFolderID, Me.ProjectName)
    ProjectFolderID = AddPMCommunityFolder(ClientFolderID, Me.ProjectName, "Project
Folder", MetadataDefinitionID)
    'Break Inheritance and set Permissions for project folder.
    PM.SetFolderPermissions(ProjectFolderID)
    'Make Folder is marked 'Private'.
    PM.SetFolderPrivate(ProjectFolderID)

    'Set folder template.
    SetFolderTemplate(ProjectFolderID, "dynamic.aspx")

    'Get the default taxonomy and import it.
    Try
        DefaultTaxonomy = GetDefaultTaxonomy()
```

```
 63:         TaxonomyID = Taxonomy.ImportTaxonomy(DefaultTaxonomy, "pm." & Me.ClientName & "."
& Me.ProjectName)
 64:     Catch ex As Exception
 65:         RemoveProject(ProjectFolderID)
 66:         Exit Sub
 67:     End Try
 68:
 69:     ''Add the Project (Wiki) folder  to the root of the Taxonomy so that all Wiki content
gets indexed.
 70:     'TaxonomySyncRequest.SyncIdList = ProjectFolderID
 71:     'TaxonomySyncRequest.TaxonomyId = TaxonomyID
 72:     'TaxonomySyncRequest.TaxonomyLanguage = ContentAPI.DefaultContentLanguage
 73:     'Taxonomy.AddTaxonomySyncFolder(TaxonomySyncRequest)
 74:
 75:     ' Add the Taxonomy to the Project (Wiki) Folder.
 76:     Try
 77:         FolderData = FolderAPI.GetFolder(ProjectFolderID)
 78:         FolderRequest.FolderId = ProjectFolderID
 79:         FolderRequest.BreakInheritButton = True
 80:         FolderRequest.DomainProduction = FolderData.DomainProduction
 81:         FolderRequest.DomainStaging = FolderData.DomainStaging
 82:         FolderRequest.FolderDescription = FolderData.Description
 83:         FolderRequest.MetaInherited = 0
 84:         FolderRequest.FolderName = FolderData.Name
 85:         FolderRequest.IsDomainFolder = FolderData.IsDomainFolder
 86:         FolderRequest.ParentId = FolderData.ParentId
 87:         FolderRequest.PublishActive = FolderData.PublishHtmlActive
 88:         FolderRequest.SiteMapPath = FolderData.SitemapPath
 89:         FolderRequest.SiteMapPathInherit = False
 90:         FolderRequest.StyleSheet = FolderData.StyleSheet
 91:         FolderRequest.TemplateFileName = FolderData.TemplateFileName
 92:         FolderRequest.XmlInherited = FolderData.XmlInherited
 93:         FolderRequest.TaxonomyInherited = False
 94:         FolderRequest.CategoryRequired = True
 95:         FolderRequest.TaxonomyInheritedFrom = ProjectFolderID
 96:         FolderRequest.TaxonomyIdList = TaxonomyID.ToString
 97:         FolderAPI.UpdateFolder(FolderRequest)
 98:     Catch ex As Exception
 99:         RemoveProject(ProjectFolderID)
100:         Exit Sub
101:     End Try
102:
103:
104:     'Add default Wiki entry.
105:     Try
106:         DefaultWikiContentID = ContentAPI.AddContent(Me.ProjectName & " Wiki", "Default
Wiki Entry", DefaultWikiContentText(), "", "<br/><!-- Wiki Summary -->",
ContentAPI.DefaultContentLanguage, ProjectFolderID, "", "", "<metadata><meta id=" & """ &
MetadataDefinitionID & """ & ">Yes</meta></metadata>")
107:     Catch ex As Exception
108:         RemoveProject(ProjectFolderID)
109:         Exit Sub
110:     End Try
111:
112:     ' Add default Wiki entry to Taxonomy "general" category.
113:     Try
114:         TaxonomyPath = "\pm." & Me.ClientName & "." & Me.ProjectName & "\General"
115:         TaxonomyCategoryID = Taxonomy.GetTaxonomyIdByPath(TaxonomyPath)
116:         TaxonomyContentRequest.ContentId = DefaultWikiContentID
117:         TaxonomyContentRequest.TaxonomyList = Convert.ToString(TaxonomyCategoryID)
118:         ContentAPI.AddTaxonomyItem(TaxonomyContentRequest)
119:     Catch ex As Exception
120:         RemoveProject(ProjectFolderID)
121:         Exit Sub
122:     End Try
123:
```

```
124:     '--------------------------------------------------------------------------------
125:
126:
127:     '- Documents Folder -
128:     '--------------------------------------------------------------------------------
129:     Try
130:         'Add Project folder
131:         DocumentsFolderID = AddPMCommunityFolder(ProjectFolderID, "Documents", "Project
Documents", -1)
132:         'Break Inheritance and set Permissions for project folder.
133:         PM.SetFolderPermissions(DocumentsFolderID)
134:         'Make Folder is marked 'Private'.
135:         PM.SetFolderPrivate(DocumentsFolderID)
136:         'Set folder template.
137:         SetFolderTemplate(DocumentsFolderID, "Documents.aspx")
138:     Catch ex As Exception
139:         RemoveProject(ProjectFolderID)
140:         Exit Sub
141:     End Try
142:
143:     '--------------------------------------------------------------------------------
144:
145:
146:     '- Milestones Folder -
147:     '--------------------------------------------------------------------------------
148:     Try
149:         'Get Milestones SmartForm ID.
150:         MilestonesSmartFormID = PM.GetSmartFormID()
151:         'Add Project folder.
152:         MilestonesFolderID = AddPMCommunityFolder(ProjectFolderID, "Milestones", "Project
Milestones", -1, "StarterApps/ProjectManagement/Milestones.aspx", MilestonesSmartFormID)
153:         'Break Inheritance and set Permissions for project folder.
154:         PM.SetFolderPermissions(MilestonesFolderID)
155:         'Make Folder is marked 'Private'.
156:         PM.SetFolderPrivate(MilestonesFolderID)
157:         'Set Read-Only Permissions on this folder for Membersip users (membership users
cannot add content via smartforms).
158:         MakeFolderReadOnly(MilestonesFolderID)
159:         'Set folder template.
160:         'SetFolderTemplate(MilestonesFolderID, "Milestones.aspx")
161:
162:     Catch ex As Exception
163:         RemoveProject(ProjectFolderID)
164:         Exit Sub
165:     End Try
166:
167:
168:     ''Add Milestones SmartForm to Milestones Folder.
169:     'SmartFormCollection.Add(MilestonesSmartFormID, MilestonesSmartFormID)
170:     'ContentRW.UpdateFolderXmlList(MilestonesFolderID, MilestonesSmartFormID,
SmartFormCollection)
171:     '--------------------------------------------------------------------------------
172:
173:     '- Membership User's Blog -
174:     '--------------------------------------------------------------------------------
175:     Try
176:         'Add Project Blog.
177:         BlogID = AddBlog(ProjectFolderID, "Blog", Me.ProjectName & " Blog", "All client
members may view and contribute to this blog")
178:         'Set folder template.
179:         SetFolderTemplate(BlogID, "Blog.aspx")
180:     Catch ex As Exception
181:         RemoveProject(ProjectFolderID)
182:         Exit Sub
183:     End Try
184:
```

```
185:     'Assign "Blog" metadata definition to blog.
186:     Try
187:         MetadataDefinitionID =
PM.GetMetadataDefinitionID(ProjectManagement.MetadataDefinitionTypes.MembershipBlog)
188:         AddMetadataDefinitionToFolder(BlogID, MetadataDefinitionID)
189:     Catch ex As Exception
190:         RemoveProject(ProjectFolderID)
191:         Exit Sub
192:     End Try
193:
194:
195:     'This blog has inherited the Taxonomy of the Wiki (parent folder).
196:     'Remove Taxonomy inheritance, the Taxonomy itself, and the Taxonomy category
requirement from this folder (blog).
197:     'This code may be removed at some point.  As of build 99 taxonomies are inherited.
198:     'FolderData = FolderAPI.GetFolder(BlogID)
199:     'FolderRequest.FolderId = BlogID
200:     'FolderRequest.BreakInheritButton = True
201:     'FolderRequest.DomainProduction = FolderData.DomainProduction
202:     'FolderRequest.DomainStaging = FolderData.DomainStaging
203:     'FolderRequest.FolderDescription = FolderData.Description
204:     'FolderRequest.MetaInherited = 0
205:     'FolderRequest.FolderName = FolderData.Name
206:     'FolderRequest.IsDomainFolder = FolderData.IsDomainFolder
207:     'FolderRequest.ParentId = FolderData.ParentId
208:     'FolderRequest.PublishActive = FolderData.PublishHtmlActive
209:     'FolderRequest.SiteMapPath = FolderData.SitemapPath
210:     'FolderRequest.SiteMapPathInherit = False
211:     'FolderRequest.StyleSheet = FolderData.StyleSheet
212:     'FolderRequest.TemplateFileName = FolderData.TemplateFileName
213:     'FolderRequest.XmlInherited = FolderData.XmlInherited
214:     'FolderRequest.TaxonomyInherited = False
215:     'FolderRequest.CategoryRequired = False
216:     'FolderRequest.TaxonomyInheritedFrom = BlogID
217:     'FolderRequest.TaxonomyIdList = ""
218:     'FolderAPI.UpdateFolder(FolderRequest)
219:
220:     'Add a default blog entry (Lorem Ipsum).
221:     Try
222:         AddDefaultBlogEntry(BlogID, Me.ProjectName & " Blog")
223:     Catch ex As Exception
224:         RemoveProject(ProjectFolderID)
225:         Exit Sub
226:     End Try
227:
228:     '-------------------------------------------------------------------------------
229:
230:     '- CMS User's Blog -
231:     '-------------------------------------------------------------------------------
232:     Try
233:         'Add Private Blog.
234:         BlogID = AddBlog(ProjectFolderID, "Private Blog", Me.ProjectName & " Private
Blog", "Only CMS users may view and contribute to this blog")
235:         'Remove Membership user group from CMS User's Blog.
236:         UserGroupData = UserGroup.GetUserGroupByName("Starterapps.pm." & Me.ClientName)
237:         Permissions.DeleteItemPermission(UserGroupData.GroupId, True, BlogID,
EkEnumeration.CMSObjectTypes.Folder)
238:         'Set folder template.
239:         SetFolderTemplate(BlogID, "Blog.aspx")
240:
241:     Catch ex As Exception
242:         RemoveProject(ProjectFolderID)
243:         Exit Sub
244:     End Try
245:
246:     'Assign "PrivateBlog" metadata definition to blog.
```

```
247:     Try
248:         MetadataDefinitionID =
PM.GetMetadataDefinitionID(ProjectManagement.MetadataDefinitionTypes.CMSBlog)
249:         AddMetadataDefinitionToFolder(BlogID, MetadataDefinitionID)
250:     Catch ex As Exception
251:         RemoveProject(ProjectFolderID)
252:         Exit Sub
253:     End Try
254:
255:
256:     'This blog has inherited the Taxonomy of the Wiki (parent folder).
257:     'Remove Taxonomy inheritance, the Taxonomy itself, and the Taxonomy category
requirement from this folder (blog).
258:     'This code may be removed at some point.  As of build 99 taxonomies are inherited.
259:     'FolderData = FolderAPI.GetFolder(BlogID)
260:     'FolderRequest.FolderId = BlogID
261:     'FolderRequest.BreakInheritButton = True
262:     'FolderRequest.DomainProduction = FolderData.DomainProduction
263:     'FolderRequest.DomainStaging = FolderData.DomainStaging
264:     'FolderRequest.FolderDescription = FolderData.Description
265:     'FolderRequest.MetaInherited = 0
266:     'FolderRequest.FolderName = FolderData.Name
267:     'FolderRequest.IsDomainFolder = FolderData.IsDomainFolder
268:     'FolderRequest.ParentId = FolderData.ParentId
269:     'FolderRequest.PublishActive = FolderData.PublishHtmlActive
270:     'FolderRequest.SiteMapPath = FolderData.SitemapPath
271:     'FolderRequest.SiteMapPathInherit = False
272:     'FolderRequest.StyleSheet = FolderData.StyleSheet
273:     'FolderRequest.TemplateFileName = FolderData.TemplateFileName
274:     'FolderRequest.XmlInherited = FolderData.XmlInherited
275:     ''FolderRequest.TaxonomyInherited = False
276:     ''FolderRequest.CategoryRequired = False
277:     ''FolderRequest.TaxonomyInheritedFrom = BlogID
278:     ''FolderRequest.TaxonomyIdList = ""
279:     'FolderAPI.UpdateFolder(FolderRequest)
280:
281:     'Add a default blog entry (Lorem Ipsum).
282:     Try
283:         AddDefaultBlogEntry(BlogID, Me.ProjectName & " Private Blog")
284:
285:     Catch ex As Exception
286:         RemoveProject(ProjectFolderID)
287:         Exit Sub
288:     End Try
289:
290:     '-------------------------------------------------------------------------------
291:
292:
293:     '- Issues Folder -
294:     '-------------------------------------------------------------------------------
295:     Try
296:         'Get Issues SmartForm ID.
297:         IssuesSmartFormID = PM.GetIssuesSmartFormID()
298:         'Add Project folder.
299:         IssuesFolderID = AddPMCommunityFolder(ProjectFolderID, "Issues", "Project
Issues", -1, "StarterApps/ProjectManagement/Issues.aspx", IssuesSmartFormID)
300:         'Break Inheritance and set Permissions for project folder.
301:         PM.SetFolderPermissions(IssuesFolderID)
302:         'Make Folder is marked 'Private'.
303:         PM.SetFolderPrivate(IssuesFolderID)
304:         'Set Read-Only Permissions on this folder for Membersip users (membership users
cannot add content via smartforms).
305:         MakeFolderReadOnly(IssuesFolderID)
306:         'Set folder template.
307:         SetFolderTemplate(IssuesFolderID, "Issues.aspx")
308:     Catch ex As Exception
```

```
309:         RemoveProject(ProjectFolderID)
310:         Exit Sub
311:     End Try
312:
313:
314:
315:     'Add Issues SmartForm to Issues Folder.
316:     Try
317:         SmartFormCollection.Add(IssuesSmartFormID, IssuesSmartFormID)
318:         'ContentRW.UpdateFolderXmlList(IssuesFolderID, IssuesSmartFormID,
SmartFormCollection)
319:     Catch ex As Exception
320:         RemoveProject(ProjectFolderID)
321:         Exit Sub
322:     End Try
323:
324:     '--------------------------------------------------------------------------------
325:
326:     '--------------------------------------------------------------------------------
327:
328:     '- Project Discussion -
329:     '--------------------------------------------------------------------------------
330:     Try
331:         DiscussionID = AddDiscussion(ProjectFolderID)
332:         'Set folder template.
333:         SetFolderTemplate(DiscussionID, "Discussion.aspx")
334:     Catch ex As Exception
335:         RemoveProject(ProjectFolderID)
336:         Exit Sub
337:     End Try
338:
339:     '--------------------------------------------------------------------------------
340: End Sub
```

Projects.DefaultBlogContentText Method

Returns default Blog content.

C#

```
private String DefaultBlogContentText();
```

Visual Basic

```
Private Function DefaultBlogContentText() As String
```

Returns

String

Remarks

This method is used to insert content into a content block placeholder when creating a new Blog.

Body Source

```
1: Private Function DefaultBlogContentText() As String
2:     Dim text As StringBuilder = New StringBuilder()
3:     text.Append("<h3>Highlights:</h3><p> </p>")
4:     text.Append("<h3>Status:</h3><p> </p>")
5:     text.Append("<h3>Fires:</h3><p> </p>")
6:
7:     Return text.ToString()
8: End Function
```

Projects.DefaultWikiContentText Method

Returns default Wiki (see page 130) content.

C#

```
private String DefaultWikiContentText();
```

Visual Basic

```
Private Function DefaultWikiContentText() As String
```

Returns

String

Remarks

This method is used to insert content into a content block placeholder when creating a new Wiki (see page 130).

Body Source

```
1: Private Function DefaultWikiContentText() As String
2:     Dim text As StringBuilder = New StringBuilder()
3:     text.Append("<h3>Stake Holder:</h3><p> </p>")
4:     text.Append("<h3>Project Goals:</h3><p> </p>")
5:     text.Append("<h3>Success Criteria:</h3><p> </p>")
6:
7:     Return text.ToString()
8: End Function
```

Projects.GetAllProjects Method

This method retrieves all project folder names and IDs that the user has permissions to view.

C#

```
public NameValueCollection GetAllProjects();
```

Visual Basic

```
Public Function GetAllProjects() As NameValueCollection
```

Returns

A NameValueCollection made up of the folder's name and the folder's ID.

Remarks

This method respects permissions (using the GetChildFolders() method) and only returns the folders the user is allowed to view.

Body Source

```
 1: Public Function GetAllProjects() As NameValueCollection
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim PM As New Ektron.Cms.StarterApps.ProjectManagement.ProjectManagement
 4:     Dim PMFolderID As Integer
 5:     Dim Projects As New NameValueCollection
 6:     Dim FolderData() As FolderData
 7:     Dim Folder As New FolderData
 8:
 9:     'Get the Project Management folder's ID.
10:     PMFolderID = PM.Initialize()
11:     'Create nameValueCollection of folder names and folder IDs.
12:     FolderData = FolderAPI.GetChildFolders(PMFolderID, False)
13:     If Not FolderData Is Nothing Then
14:         For Each Folder In FolderData
15:             Projects.Add(Folder.Name, Folder.Id)
16:         Next
17:     End If
18:     'Return the NameValueCollection.
19:     Return Projects
20: End Function
```

Projects.GetDefaultTaxonomy Method

This method loads the default taxonomy ("DefaultProjectTaxonomy.xml") into a string.

C#

```
private String GetDefaultTaxonomy();
```

Visual Basic

```
Private Function GetDefaultTaxonomy() As String
```

Returns

An Taxonomy XML String.

Remarks

This method is used during project creation. The LoadTaxonomy() method takes an xml string as a param. We grab the default taxonomy, load it as a System.Xml.XmlDocument object, and return InnerXML.

Body Source

```
1: Private Function GetDefaultTaxonomy() As String
2:     Dim DefaultTaxonomy As New System.Xml.XmlDocument
3:
4:
DefaultTaxonomy.Load(System.Web.HttpContext.Current.Server.MapPath("../xml/DefaultProjectTaxono
my.xml"))
5:     GetDefaultTaxonomy = DefaultTaxonomy.InnerXml
6:
7:     Return GetDefaultTaxonomy
8: End Function
```

Projects.GetProjectTaxonomyID Method

Gets the ID of the taxonomy associated with the project folder.

C#

```
public int GetProjectTaxonomyID(int ProjectFolderID);
```

Visual Basic

```
Public Function GetProjectTaxonomyID(ByVal ProjectFolderID As Integer) As Integer
```

Parameters

Parameters	Description
ProjectFolderID	FolderID of the project folder.

Returns

Integer

Remarks

This method returns the taxonomy ID associated with the Project ("Wiki (see page 130)").

Body Source

```
1: Public Function GetProjectTaxonomyID(ByVal ProjectFolderID As Integer) As Integer
2:     Dim FolderAPI As New Ektron.Cms.API.Folder
3:     Dim FolderData As New Ektron.Cms.FolderData
4:     Dim Taxonomy As New Ektron.Cms.API.Content.Taxonomy
5:     Dim TaxonomyBaseData() As Ektron.Cms.TaxonomyBaseData
6:     Dim TaxonomyBaseDatum As New Ektron.Cms.TaxonomyBaseData
7:     Dim ProjectName, ClientName As String
8:     Dim TaxonomyID As Integer = -1
9:
```

```
10:     '    'Get the project's name.
11:     FolderData = FolderAPI.GetFolder(ProjectFolderID, True)
12:     ProjectName = System.Web.HttpUtility.HtmlDecode(FolderData.Name)
13:     TaxonomyBaseData = FolderData.FolderTaxonomy
14:
15:     'Get the client's name.
16:     FolderData = FolderAPI.GetFolder(FolderData.ParentId)
17:     ClientName = System.Web.HttpUtility.HtmlDecode(FolderData.Name)
18:
19:     'Get the project's taxonomy.
20:     For Each TaxonomyBaseDatum In TaxonomyBaseData
21:         If TaxonomyBaseDatum.TaxonomyName = "pm." & ClientName & "." & ProjectName Then
22:             TaxonomyID = TaxonomyBaseDatum.TaxonomyId
23:             Exit For
24:         End If
25:     Next
26:
27:     Return TaxonomyID
28: End Function
```

Projects.LoremIpsumLong Method

This method generates placeholder text.

C#

```
private String LoremIpsumLong();
```

Visual Basic

```
Private Function LoremIpsumLong() As String
```

Returns

3 html paragraphs of Lorem Ipsum text.

Remarks

This is here to save the hassle of cutting and pasting placeholder text.

Body Source

```
1: Private Function LoremIpsumLong() As String
2:     LoremIpsumLong = "<p>" & "Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Donec pharetra, nisl tristique semper rutrum, risus magna congue metus, faucibus venenatis
orci metus vel lectus. Duis at ante. Donec bibendum. In lectus orci, rhoncus hendrerit, cursus
dictum, auctor pharetra, tortor. Maecenas metus mi, mollis non, iaculis eget, ornare in,
risus. Cras ut odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere
cubilia Curae; Vestibulum eu lorem. Suspendisse quis mauris. Quisque nisl. Maecenas felis.
Fusce dui odio, tristique sit amet, tincidunt et, volutpat a, diam. Phasellus non libero.
Phasellus et leo a libero cursus congue. Pellentesque feugiat est nec nulla." & "</p>"
3:     LoremIpsumLong = LoremIpsumLong & "<p>" & "Proin et enim interdum nisl elementum
porttitor. Etiam quam turpis, hendrerit quis, dictum id, ultricies quis, tortor. Vestibulum
ante metus, aliquet quis, posuere at, pharetra at, lorem. Etiam nec felis. Integer fringilla
bibendum arcu. Pellentesque iaculis libero vitae libero. Aenean euismod mollis lorem. Etiam
sit amet arcu. Nunc nisi dolor, luctus vel, rhoncus commodo, tempor ut, turpis. Praesent
auctor vehicula erat. Sed porttitor accumsan massa. Phasellus sed lectus." & "</p>"
4:     LoremIpsumLong = LoremIpsumLong & "<p>" & "Nunc porta ornare pede. Donec tincidunt
elementum mauris. Aliquam interdum elit id neque. Donec quis eros ac tellus nonummy laoreet.
Nam ultrices, risus sit amet molestie dignissim, dolor turpis tempus est, in semper dui risus
vel nulla. Morbi iaculis metus at ipsum. Praesent lectus pede, malesuada eget, bibendum quis,
bibendum luctus, orci. Pellentesque rhoncus lacus a mi. Nam vulputate lorem eget neque.
Aliquam aliquet purus in erat. Vivamus dignissim fringilla enim. Curabitur id magna. Maecenas
aliquet posuere ante. Proin arcu tortor, cursus in, facilisis quis, sagittis sit amet, est.
Nunc et erat at ipsum semper ultricies. Sed mi. Vestibulum sed nisi. Donec massa nisl, posuere
sed, pharetra et, egestas vitae, magna." & "</p>"
5:     Return LoremIpsumLong
6: End Function
```

Projects.LoremIpsumShort Method

This method generates placeholder text.

C#

```
private String LoremIpsumShort();
```

Visual Basic

```
Private Function LoremIpsumShort() As String
```

Returns

1 html paragraphs of Lorem Ipsum text.

Remarks

This is here to save the hassle of cutting and pasting placeholder text.

Body Source

```
1: Private Function LoremIpsumShort() As String
2:     LoremIpsumShort = "<p>" & "Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Donec pharetra, nisl tristique semper rutrum, risus magna congue metus, faucibus venenatis
orci metus vel lectus. Duis at ante. Donec bibendum. In lectus orci, rhoncus hendrerit, cursus
dictum, auctor pharetra, tortor. Maecenas metus mi, mollis non, iaculis eget, ornare in,
risus. Cras ut odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere
cubilia Curae; Vestibulum eu lorem. Suspendisse quis mauris. Quisque nisl. Maecenas felis.
Fusce dui odio, tristique sit amet, tincidunt et, volutpat a, diam. Phasellus non libero.
Phasellus et leo a libero cursus congue. Pellentesque feugiat est nec nulla." & "</p>"
3:     Return LoremIpsumShort
4: End Function
```

Projects.MakeFolderNameUnique Method

This method ensures a project name is unique.

C#

```
private String MakeFolderNameUnique(int ParentFolderID, String NewFolderName);
```

Visual Basic

```
Private Function MakeFolderNameUnique(ByVal ParentFolderID As Integer, ByVal NewFolderName As
String) As String
```

Parameters

Parameters	Description
ParentFolderID	Folder ID of the parent folder (the "client" folder).
NewFolderName	The desired name of the child folder (the "project" folder).

Returns

The unique name.

Remarks

This method checks to see if there is a sibling folder that has already been given the name passed in the parameter "NewFolderName." If a folder already exists with this name, the function appends to the name a *count* like Windows does.

For example, If NewFolderName = "Ektron" and a folder named "Ektron" already exists, this method returns "Ektron (1)."

Body Source

```
 1: Private Function MakeFolderNameUnique(ByVal ParentFolderID As Integer, ByVal NewFolderName
As String) As String
 2:     Dim UniqueName As String = NewFolderName
 3:     Dim IsUnique As Boolean = True
```

```
    Dim i As Integer = 1
    Dim currentComponentName As String
    Dim FolderAPI As New Ektron.Cms.API.Folder

    Dim SiblingFolderData() As FolderData
    Dim SiblingFolder As FolderData

    SiblingFolderData = FolderAPI.GetChildFolders(ParentFolderID)
    If Not SiblingFolderData Is Nothing Then
        For Each SiblingFolder In SiblingFolderData
            If SiblingFolder.Name = NewFolderName Then
                IsUnique = False
            End If
        Next
    End If

    If IsUnique = False Then
        For Each SiblingFolder In SiblingFolderData
            currentComponentName = NewFolderName & " (" & i & ")"
            If currentComponentName = SiblingFolder.Name Then
                i = i + 1
            End If
        Next
        UniqueName = NewFolderName & " (" & i & ")"
    End If

    Return UniqueName
End Function
```

Projects.MakeFolderReadOnly Method

This method is for use with the Milestones folder in the Project Management Starter App.

C#

```
private MakeFolderReadOnly(int FolderID);
```

Visual Basic

```
Private Sub MakeFolderReadOnly(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
`FolderID`	This should be the ID of the "Milestones" folder.

Remarks

The "Milestones" folder is read-only for Membership users, because to add content, you must use a Smart Form. Membership users cannot add content via Smart Forms. Thus, this folder is read-only for Membership users.

Body Source

```
Private Sub MakeFolderReadOnly(ByVal FolderID As Integer)
    Dim User As New Ektron.Cms.API.User.User
    Dim UserGroup As New Ektron.Cms.UserGroupData
    Dim Users As New Ektron.Cms.StarterApps.ProjectManagement.Users
    Dim MembershipGroupID As Integer
    Dim UserPermissionData As New Ektron.Cms.UserPermissionData
    Dim Permissions As New Ektron.Cms.API.Permissions

    'Get the Membership user group's ID.
    UserGroup = User.GetUserGroupByName("Starterapps.pm." & Me.ClientName)
    MembershipGroupID = UserGroup.GroupId

    'Set Read-Only permissions.
    UserPermissionData.FolderId = FolderID
    UserPermissionData.UserId = -1
```

```
16:     UserPermissionData.GroupId = MembershipGroupID
17:     UserPermissionData.IsReadOnly = True
18:     UserPermissionData.CanAdd = False
19:     UserPermissionData.CanEdit = False
20:     UserPermissionData.IsReadOnlyLib = True
21:     UserPermissionData.CanAddToImageLib = False
22:     UserPermissionData.CanAddToFileLib = False
23:     'Update the folder.
24:     Try
25:         Permissions.UpdateItemPermission(UserPermissionData)
26:     Catch ex As Exception
27:
28:     End Try
29:
30: End Sub
```

Projects.RemoveProject Method

This method removes a project from the Project Management StarterAPP.

C#

```
public RemoveProject(int _ProjectFolderID);
```

Visual Basic

```
Public Sub RemoveProject(ByVal _ProjectFolderID As Integer)
```

Parameters

Parameters	Description
ProjectFolderID	The ID of the project (folder ID) that you wish to remove.

Remarks

This method removes a project and all project contents including Wiki (see page 130) content (and the associated Wiki (see page 130) taxonomy), discussion content, documents and blogs.

Body Source

```
 1: Public Sub RemoveProject(ByVal _ProjectFolderID As Integer)
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim FolderData As New Ektron.Cms.FolderData
 4:     Dim ProjectName, ClientName As String
 5:     Dim Taxonomy As New Ektron.Cms.API.Content.Taxonomy
 6:     Dim TaxonomyBaseData() As Ektron.Cms.TaxonomyBaseData
 7:     Dim TaxonomyBaseDatum As New Ektron.Cms.TaxonomyBaseData
 8:     Dim TaxonomyRequest As New Ektron.Cms.TaxonomyRequest
 9:
10:     'Get the project's name.
11:     FolderData = FolderAPI.GetFolder(_ProjectFolderID, True)
12:     ProjectName = FolderData.Name
13:     TaxonomyBaseData = FolderData.FolderTaxonomy
14:
15:     'Get the client's name.
16:     FolderData = FolderAPI.GetFolder(FolderData.ParentId)
17:     ClientName = FolderData.Name
18:
19:     'Delete the project's taxonomy.
20:     For Each TaxonomyBaseDatum In TaxonomyBaseData
21:         If TaxonomyBaseDatum.TaxonomyName = "pm." &
Ektron.Cms.Common.EkFunctions.HtmlDecode(ClientName) & "." &
Ektron.Cms.Common.EkFunctions.HtmlDecode(ProjectName) Then
22:             TaxonomyRequest.TaxonomyId = TaxonomyBaseDatum.TaxonomyId
23:             TaxonomyRequest.TaxonomyLanguage = FolderAPI.DefaultContentLanguage
24:             Taxonomy.DeleteTaxonomy(TaxonomyRequest)
25:             Exit For
26:         End If
```

```
27:     Next
28:
29:     'Delete the project's folder.
30:     Try
31:         FolderAPI.DeleteFolderById(_ProjectFolderID)
32:     Catch ex As Exception
33:         Throw New Exception("Cannot Delete Project Folder: " & ex.Message)
34:     End Try
35: End Sub
```

Projects.SetFolderTemplate Method

This method sets the default .aspx template for a folder.

C#

```
public SetFolderTemplate(int FolderID, String TemplateName);
```

Visual Basic

```
Public Sub SetFolderTemplate(ByVal FolderID As Integer, ByVal TemplateName As String)
```

Parameters

Parameters	Description
FolderID	The ID of the folder to which the template is assigned.
TemplateName	The name of the template to assign to the folder.

Remarks

In addition to assigning the template specified in the parameter, this method also sets the folder's associated stylesheet to "nothing.css." This is done to ensure all css styles are applied via the css files (and styles) specified in the html head section. That is, so unwanted styles don't sneak in uninvited.

Body Source

```
 1: Public Sub SetFolderTemplate(ByVal FolderID As Integer, ByVal TemplateName As String)
 2:     Dim FolderAPI As New Ektron.Cms.API.Folder
 3:     Dim FolderData As New Ektron.Cms.FolderData
 4:     Dim FolderRequest As New Ektron.Cms.FolderRequest
 5:
 6:     FolderData = FolderAPI.GetFolder(FolderID, True)
 7:     FolderRequest.FolderId = FolderID
 8:     FolderRequest.BreakInheritButton = True
 9:     FolderRequest.DomainProduction = FolderData.DomainProduction
10:     FolderRequest.DomainStaging = FolderData.DomainStaging
11:     FolderRequest.FolderDescription = FolderData.Description
12:     FolderRequest.MetaInherited = 0
13:     FolderRequest.FolderName = FolderData.Name
14:     FolderRequest.IsDomainFolder = FolderData.IsDomainFolder
15:     FolderRequest.ParentId = FolderData.ParentId
16:     FolderRequest.PublishActive = FolderData.PublishHtmlActive
17:     FolderRequest.SiteMapPath = FolderData.SitemapPath
18:     FolderRequest.SiteMapPathInherit = False
19:     FolderRequest.StyleSheet = "StarterApps/ProjectManagement/css/ewebeditprostyles.css"
20:     FolderRequest.TemplateFileName = "StarterApps/ProjectManagement/" & TemplateName
21:     FolderRequest.XmlInherited = FolderData.XmlInherited
22:     FolderAPI.UpdateFolder(FolderRequest)
23: End Sub
```

Projects Properties

The properties of the Projects class are listed here.

Public Properties

	Name	Description
	ClientID (see page 114)	The ID of the folder associated with the selected client.

	ClientName (see page 114)	The name of the selected client as an unescaped string.
	PMFolderID (see page 114)	The Integer ID of the project management folder.
	ProjectName (see page 114)	The ProjectName property is the name associated with the Project Folder.

Legend

	Property

Projects.ClientID Property

The ID of the folder associated with the selected client.

C#

```
public int ClientID;
```

Visual Basic

```
Public Property ClientID() As Integer
```

Returns

The FolderID of the selected client.

Projects.ClientName Property

The name of the selected client as an unescaped string.

C#

```
public String ClientName;
```

Visual Basic

```
Public Property ClientName() As String
```

Returns

The name of the client.

Projects.PMFolderID Property

The Integer ID of the project management folder.

C#

```
public int PMFolderID;
```

Visual Basic

```
Public Property PMFolderID() As Integer
```

Returns

The ID of the PM Folder.

Projects.ProjectName Property

The ProjectName property is the name associated with the Project Folder.

C#

```
public String ProjectName;
```

Visual Basic

```
Public Property ProjectName() As String
```

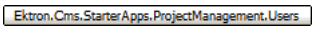
Returns

Returns unescaped project name.

Users Class

Contains methods for managing users in the Project Manager Starter Application.

Class Hierarchy



C#

```
public class Users;
```

Visual Basic

```
Public Class Users
```

File

ProjectManagement.vb

Users Methods

The methods of the Users class are listed here.

Public Methods

	Name	Description
	AddCMSUserGroup (see page 115)	This method adds a CMS (see page 1) user group to the Project Management StarterApp.
	AddMembershipUserGroup (see page 116)	This method adds a Membership user group to the Project Management StarterApp.
	AddMembershipUserGroupToFolder (see page 117)	This method adds a Membership user group to a folder and sets the permissions appropriate to the Project Management Starer App for the group on the folder.
	AddMemberToClient (see page 118)	This method adds a user (by user ID) to a user group (by user group ID).
	AddUserGroupToFolder (see page 119)	This method adds a user group to a folder and sets Permissions for the group on the folder.
	AddUserToGroup (see page 120)	This method adds a user (by user ID) to a user group (by group name).
	RemoveMemberFromClient (see page 121)	This method removes a Membership user from a client user group.
	RemoveUserGroup (see page 121)	This method removes a user group (by user group name).

Legend

	Method

Users.AddCMSUserGroup Method

This method adds a CMS (see page 1) user group to the Project Management StarterApp.

C#

```
public int AddCMSUserGroup(String CMSGroupName);
```

Visual Basic

```
Public Function AddCMSUserGroup(ByVal CMSGroupName As String) As Integer
```

Parameters

Parameters	Description
CMSGroupName	The name of the CMS (see page 1) group to add.

Returns

The ID of the newly created CMS (see page 1) user group.

Remarks

This method is meant to be executed only when the StarterApp initializes itself. There is only 1 CMS (see page 1) user group in the StarterApp by design. All members of the CMS (see page 1) user group have permissions to do everything to all folders within the StarterApp.

Body Source

```
 1: Public Function AddCMSUserGroup(ByVal CMSGroupName As String) As Integer
 2:     Dim User As New Ektron.Cms.API.User.User
 3:     Dim UserGroups() As UserGroupData
 4:     Dim UserGroup As New UserGroupData
 5:     Dim UserGroupExists As Boolean = False
 6:     Dim CMSGroupID As Integer
 7:
 8:     'See if the CMS user group name already exists.
 9:     UserGroups = User.GetAllUserGroups(EkEnumeration.UserTypes.AuthorType)
10:     For Each UserGroup In UserGroups
11:         If UserGroup.GroupName = CMSGroupName Then
12:             CMSGroupID = UserGroup.GroupId
13:             UserGroupExists = True
14:             Exit For
15:         End If
16:     Next
17:
18:     'If the CMS user group doesn't exist, create it.
19:     If UserGroupExists = False Then
20:         User.AddUserGroup(CMSGroupName)
21:         UserGroup = User.GetUserGroupByName(CMSGroupName)
22:         CMSGroupID = UserGroup.GroupId
23:     End If
24:
25:     'return the CMS user group ID.
26:     Return CMSGroupID
27: End Function
```

Users.AddMembershipUserGroup Method

This method adds a Membership user group to the Project Management StarterApp.

C#

```
public int AddMembershipUserGroup(String MemershipGroupName);
```

Visual Basic

```
Public Function AddMembershipUserGroup(ByVal MemershipGroupName As String) As Integer
```

Parameters

Parameters	Description
MemershipGroupName	The name of the Membership user group to add.

Returns

The ID of the newly created Membership user group.

Remarks

This method is executed once per client. That is, each client gets its own Membership user group. The client's Membership user group is deleted when the client is removed with the RemoveClient() method.

Body Source

```
1: Public Function AddMembershipUserGroup(ByVal MemershipGroupName As String) As Integer
```

```
 2:     Dim MembershipUserGroupCollection As New Collection
 3:     Dim User As New Ektron.Cms.API.User.User
 4:     Dim UserGroups() As UserGroupData
 5:     Dim UserGroup As New UserGroupData
 6:     Dim UserGroupExists As Boolean = False
 7:     Dim MembershipGroupID As Integer
 8:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 9:     Dim ErrorMessage As String = ""
10:
11:     'See if the CMS user group name already exists.
12:     UserGroups = User.GetAllUserGroups(EkEnumeration.UserTypes.AuthorType)
13:     For Each UserGroup In UserGroups
14:         If UserGroup.GroupName = MemershipGroupName Then
15:             MembershipGroupID = UserGroup.GroupId
16:             UserGroupExists = True
17:             Exit For
18:         End If
19:     Next
20:
21:     'If user group doesn't exist, create it.
22:     If UserGroupExists = False Then
23:         MembershipUserGroupCollection.Add(MemershipGroupName, "UserGroupName")
24:         MembershipUserGroupCollection.Add("StarterApp membership usergroup", "Description")
25:         Try
26:             Dim tmpID As Integer = ContentAPI.RequestInformationRef.CallerId
27:             ContentAPI.RequestInformationRef.CallerId = EkConstants.InternalAdmin
28:             ContentAPI.EkUserRef.AddMemberShipGroupV4(MembershipUserGroupCollection, "",
""")
29:             ContentAPI.RequestInformationRef.CallerId = tmpID
30:
31:             UserGroup = User.GetUserGroupByName(MemershipGroupName)
32:             MembershipGroupID = UserGroup.GroupId
33:         Catch ex As Exception
34:             MembershipGroupID = -1
35:             ErrorMessage = ErrorMessage & "Membership Group Could Not Be Created.  "
36:             ErrorMessage = ErrorMessage & "You may manually work around this error by
creating a Membership usergroup in the Workarea.  "
37:             ErrorMessage = ErrorMessage & "The Membership usergroup name must be '" &
MemershipGroupName & "'."
38:             Throw New Exception(ErrorMessage & ex.Message)
39:         End Try
40:
41:     End If
42:
43:     'return the user group ID.
44:     Return MembershipGroupID
45: End Function
```

Users.AddMembershipUserGroupToFolder Method

This method adds a Membership user group to a folder and sets the permissions appropriate to the Project Management Starer App for the group on the folder.

C#

```
public AddMembershipUserGroupToFolder(int FolderID, int GroupID);
```

Visual Basic

```
Public Sub AddMembershipUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer)
```

Parameters

Parameters	Description
FolderID	The folder to which you wish to assign the user group.
GroupID	The group ID of the Membership user group you wish to assign to the folder.

Remarks

This method includes the permission settings appropriate to Membership users (see page 115) in the Project Management Starter App.

Body Source

```
 1: Public Sub AddMembershipUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As
Integer)
 2:     Dim UserPermissionData As New Ektron.Cms.UserPermissionData
 3:     Dim Permissions As New Ektron.Cms.API.Permissions
 4:
 5:     'Add permissions for GroupID to FolderID.
 6:     UserPermissionData.GroupId = GroupID
 7:     UserPermissionData.FolderId = FolderID
 8:     'File/folder permissions.
 9:     UserPermissionData.CanAdd = True
10:     UserPermissionData.CanEdit = True
11:     UserPermissionData.CanDelete = False
12:     'Library permissions.
13:     UserPermissionData.CanAddToFileLib = True
14:     UserPermissionData.CanAddToImageLib = True
15:     UserPermissionData.CanAddToHyperlinkLib = False
16:     UserPermissionData.CanOverwriteLib = False
17:     'Membership users can traverse folders by default (hardcoded to True).
18:     'We are "adding" here to show explicitly that this permission setting is required.
19:     UserPermissionData.CanTraverseFolders = True
20:     Try
21:         Permissions.AddItemPermission(UserPermissionData)
22:     Catch ex As Exception
23:
24:     End Try
25:
26: End Sub
```

Users.AddMemberToClient Method

This method adds a user (by user ID) to a user group (by user group ID).

C#

```
public AddMemberToClient();
```

Visual Basic

```
Public Sub AddMemberToClient()
```

Remarks

This method requires population of two Users (see page 115) properties...

1. MemberID (see page 122)
2. ClientID (see page 122)

This method is more or less the same thing as AddUserToGroup (see page 120)() only this method takes a user group ID whereas AddUserToGroup (see page 120)() takes a user group name.

Body Source

```
 1: Public Sub AddMemberToClient()
 2:     Dim Member As New Ektron.Cms.API.User.User
 3:     Dim GroupInfo As New UserGroupData
 4:     Dim Members As New Ektron.Cms.API.User.User
 5:     Dim FolderAPI As New Ektron.Cms.API.Folder
 6:
 7:     GroupInfo = Member.GetUserGroupByName("Starterapps.pm." &
FolderAPI.GetFolder(Me.ClientID).Name)
 8:     Try
```

```
 9:         Members.AddUserToGroup(Me.MemberID, GroupInfo.GroupId)
10:     Catch ex As Exception
11:         'Do Nothing.
12:     End Try
13: End Sub
```

Users.AddUserGroupToFolder Method

This method adds a user group to a folder and sets Permissions for the group on the folder.

C#

```
public AddUserGroupToFolder(int FolderID, int GroupID, Boolean TraverseOnly);
```

Visual Basic

```
Public Sub AddUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer, ByVal
TraverseOnly As Boolean)
```

Parameters

Parameters	Description
FolderID	The folder to which to assign the user group.
GroupID	The group ID of the user group to assign to the folder.
TraverseOnly	**True** - the group will be given Traverse permissions only. **False** - the group is given all permissions.

Remarks

This method can be used as a general-purpose method. It will work against any folder not just the ones in the StarterApp. Traverse-only is meaningful only to CMS (see page 1) user groups. Membership user groups always have traverse permissions set to True (this is hard coded in the CMS (see page 1)).

Body Source

```
 1: Public Sub AddUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer, ByVal
TraverseOnly As Boolean)
 2:     Dim UserPermissionData As New Ektron.Cms.UserPermissionData
 3:     Dim Permissions As New Ektron.Cms.API.Permissions
 4:
 5:     'Add permissions for GroupID to FolderID.
 6:     UserPermissionData.GroupId = GroupID
 7:     UserPermissionData.FolderId = FolderID
 8:
 9:     If TraverseOnly = False Then
10:         'Grant everything.
11:         UserPermissionData.CanAddToFileLib = True
12:         UserPermissionData.CanOverwriteLib = True
13:         UserPermissionData.CanAddToHyperlinkLib = True
14:         UserPermissionData.CanAddToImageLib = True
15:         UserPermissionData.CanAddToQuicklinkLib = True
16:         UserPermissionData.CanApprove = True
17:         UserPermissionData.CanBreakPending = True
18:         UserPermissionData.CanCreateTask = True
19:         UserPermissionData.CanDecline = True
20:         UserPermissionData.CanAdd = True
21:         UserPermissionData.CanDelete = True
22:         UserPermissionData.CanEdit = True
23:         UserPermissionData.CanHistory = True
24:         UserPermissionData.CanPreview = True
25:         UserPermissionData.CanPublish = True
26:         UserPermissionData.CanRestore = True
27:         UserPermissionData.CanView = True
28:         UserPermissionData.CanDeleteFolders = True
29:         UserPermissionData.CanAddFolders = True
30:         UserPermissionData.CanEditFolders = True
31:         UserPermissionData.CanEditCollections = True
32:         UserPermissionData.CanTraverseFolders = True
```

```
33:     Else
34:         'Grant only Traverse and Read-Only.
35:         UserPermissionData.IsReadOnly = False
36:         UserPermissionData.IsReadOnlyLib = False
37:         UserPermissionData.CanAddToFileLib = False
38:         UserPermissionData.CanOverwriteLib = False
39:         UserPermissionData.CanAddToHyperlinkLib = False
40:         UserPermissionData.CanAddToImageLib = False
41:         UserPermissionData.CanAddToQuicklinkLib = False
42:         UserPermissionData.CanApprove = False
43:         UserPermissionData.CanBreakPending = False
44:         UserPermissionData.CanCreateTask = False
45:         UserPermissionData.CanDecline = False
46:         UserPermissionData.CanAdd = False
47:         UserPermissionData.CanDelete = False
48:         UserPermissionData.CanEdit = False
49:         UserPermissionData.CanHistory = False
50:         UserPermissionData.CanPreview = False
51:         UserPermissionData.CanPublish = False
52:         UserPermissionData.CanRestore = False
53:         UserPermissionData.CanView = False
54:         UserPermissionData.CanDeleteFolders = False
55:         UserPermissionData.CanAddFolders = False
56:         UserPermissionData.CanEditFolders = False
57:         UserPermissionData.CanEditCollections = False
58:         UserPermissionData.CanTraverseFolders = True
59:     End If
60:
61:
62:     Try
63:         Permissions.AddItemPermission(UserPermissionData)
64:     Catch ex As Exception
65:
66:     End Try
67:
68: End Sub
```

Users.AddUserToGroup Method

This method adds a user (by user ID) to a user group (by group name).

C#

```
public AddUserToGroup(int UserID, String GroupName);
```

Visual Basic

```
Public Sub AddUserToGroup(ByVal UserID As Integer, ByVal GroupName As String)
```

Parameters

Parameters	Description
UserID	The ID of the user to add to the group.
GroupName	The name of the group to which to add the user.

Remarks

This method is more or less the same thing as AddMembertoClient (see page 118)() only this method takes a user group name whereas AddMemberToClient (see page 118)() takes a user group ID.

Body Source

```
1: Public Sub AddUserToGroup(ByVal UserID As Integer, ByVal GroupName As String)
2:     Dim UserGroup As New Ektron.Cms.API.User.User
3:     Dim GroupInfo As New UserGroupData
4:     Dim Users As New Ektron.Cms.API.User.User
5:
6:     GroupInfo = UserGroup.GetUserGroupByName(GroupName)
```

```
 7:     If GroupInfo IsNot Nothing Then
 8:         Try
 9:             Users.AddUserToGroup(UserID, GroupInfo.GroupId)
10:         Catch ex As Exception
11:
12:         End Try
13:     End If
14: End Sub
```

Users.RemoveMemberFromClient Method

This method removes a Membership user from a client user group.

C#

```
public RemoveMemberFromClient();
```

Visual Basic

```
Public Sub RemoveMemberFromClient()
```

Remarks

This method requires population of two Users (see page 115) properties...

1. MemberID (see page 122)
2. ClientID (see page 122)

Body Source

```
 1: Public Sub RemoveMemberFromClient()
 2:     Dim Member As New Ektron.Cms.API.User.User
 3:     Dim GroupInfo As New UserGroupData
 4:     Dim FolderAPI As New Ektron.Cms.API.Folder
 5:
 6:     GroupInfo = Member.GetUserGroupByName("Starterapps.pm." &
FolderAPI.GetFolder(Me.ClientID).Name)
 7:     Try
 8:         Member.DeleteUserFromGroup(Me.MemberID, GroupInfo.GroupId, False)
 9:     Catch ex As Exception
10:         'Do Nothing.
11:     End Try
12: End Sub
```

Users.RemoveUserGroup Method

This method removes a user group (by user group name).

C#

```
public RemoveUserGroup(String GroupName);
```

Visual Basic

```
Public Sub RemoveUserGroup(ByVal GroupName As String)
```

Parameters

Parameters	Description
GroupName	The name of the user group to remove.

Remarks

This method is called by RemoveClient() method. When a client is removed, so is its Membership user group.

Body Source

```
 1: Public Sub RemoveUserGroup(ByVal GroupName As String)
 2:     Dim UserGroup As New API.User.User
 3:     Dim UserGroupData As New UserGroupData
```

```
 4:     Try
 5:         UserGroupData = UserGroup.GetUserGroupByName(GroupName)
 6:         UserGroup.DeleteUserGroup(UserGroupData.GroupId)
 7:     Catch ex As Exception
 8:
 9:     End Try
10: End Sub
```

Users Properties

The properties of the Users class are listed here.

Public Properties

	Name	Description
	ClientID (see page 122)	Holds the ID of the selected client.
	MemberID (see page 122)	The MemberID property is used to hold the ID of a membership user created at run-time via Registration widgets.

Legend

	Property

Users.ClientID Property

Holds the ID of the selected client.

C#

```
public int ClientID;
```

Visual Basic

```
Public Property ClientID() As Integer
```

Returns

The selected client's ID.

Users.MemberID Property

The MemberID property is used to hold the ID of a membership user created at run-time via Registration widgets.

C#

```
public int MemberID;
```

Visual Basic

```
Public Property MemberID() As Integer
```

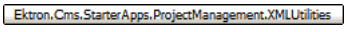
Returns

The ID of the membership user created at run-time.

XMLUtilities Class

Contains methods and properties for handling the XML used within the Project Manager Starter Application.

Class Hierarchy



C#

```
public class XMLUtilities;
```

Visual Basic

```
Public Class XMLUtilities
```

File

ProjectManagement.vb

XMLUtilities Enumerations

The enumerations of the XMLUtilities class are listed here.

Enumerations

	Name	Description
	XMLSourceTypes (see page 123)	Sets the input type of the XML to be transformed.
	XSLTSourceTypes (see page 123)	Sets the input type of the XSLT to be used as the transforming document.

Legend

	Enumeration

Ektron.Cms.StarterApps.ProjectManagement.XMLUtilities.XMLSourceTypes Enumeration

Sets the input type of the XML to be transformed.

C#

```
public enum XMLSourceTypes {
}
```

Visual Basic

```
Public Enum XMLSourceTypes
End Enum
```

File

ProjectManagement.vb

Remarks

If the type is set to XMLDocument (see page 128), you must set the XMLDocument (see page 128) property to the XMLDocument (see page 128) you wish to transform. Otherwise, set the the XMLSource (see page 128) property to the string location of the XMLDocument (see page 128) you wish to transform.

Ektron.Cms.StarterApps.ProjectManagement.XMLUtilities.XSLTSourceTypes Enumeration

Sets the input type of the XSLT to be used as the transforming document.

C#

```
public enum XSLTSourceTypes {
}
```

Visual Basic

```
Public Enum XSLTSourceTypes
End Enum
```

File

ProjectManagement.vb

Remarks

You must also set the the XSLTSource (see page 129) property to the string location of the XSLTDocument you wish to use as the transformation document.

XMLUtilities Methods

The methods of the XMLUtilities class are listed here.

Public Methods

	Name	Description
	ApplyXMLAttribute (see page 124)	This method applies XML attributes to an XML Node.
	PrettyPrintXML (see page 124)	Returns the contents of XMLDocument (see page 128) nicely indented.
	TransformXML (see page 125)	This method is a general purpose XML transformation method. This method allows arguments to be passed to the XSLT transformation.

Legend

	Method

XMLUtilities.ApplyXMLAttribute Method

This method applies XML attributes to an XML Node.

C#

```
public ApplyXMLAttribute(XmlDocument IADoc, XmlNode Node, String AttributeName, String
AttributeValue);
```

Visual Basic

```
Public Sub ApplyXMLAttribute(ByRef IADoc As XmlDocument, ByRef Node As XmlNode, ByVal
AttributeName As String, ByVal AttributeValue As String)
```

Parameters

Parameters	Description
IADoc	The XML document being parsed.
Node	The node in which to add the attribute.
AttributeName	The attribute's name.
AttributeValue	The attribute's value.

Remarks

This is a general purpose method meant to simplify adding attributes to nodes. It's meant to help keep code tidy.

Body Source

```
1: Public Sub ApplyXMLAttribute(ByRef IADoc As XmlDocument, ByRef Node As XmlNode, ByVal
AttributeName As String, ByVal AttributeValue As String)
2:     Dim Attribute As XmlAttribute
3:     Attribute = IADoc.CreateAttribute(AttributeName)
4:     Attribute.InnerText = AttributeValue
5:     Node.Attributes.Append(Attribute)
6: End Sub
```

XMLUtilities.PrettyPrintXML Method

Returns the contents of XMLDocument (see page 128) nicely indented.

C#

```
public String PrettyPrintXML(System.Xml.XmlDocument XMLDocument);
```

Visual Basic

```
Public Function PrettyPrintXML(ByRef XMLDocument As System.Xml.XmlDocument) As String
```

Parameters

Parameters	Description
XMLDocument	The XMLdocument to format.

Returns

Formatted string.

Remarks

Used primarily to make long XML/XHTML readable.

Body Source

```
 1: Public Function PrettyPrintXML(ByRef XMLDocument As System.Xml.XmlDocument) As String
 2:     'Create stream in which to load the formatted XML document.
 3:     Dim OutputStream As System.IO.Stream = New System.IO.MemoryStream
 4:
 5:     'Create XMLTextWriter with formatting specifics, and direct output into OutputStream.
 6:     Dim MyXmlTextWriter As New System.Xml.XmlTextWriter(OutputStream, Nothing)
 7:     MyXmlTextWriter.Formatting = Formatting.Indented
 8:     MyXmlTextWriter.Indentation = 5
 9:     MyXmlTextWriter.IndentChar = " "
10:
11:     'Load the contents of XMLDocument argument into XMLTextWriter.
12:     XMLDocument.Save(MyXmlTextWriter)
13:
14:     'Load Formatted XML Stream into StreamReader.
15:     Dim XMLStreamReader As New System.IO.StreamReader(OutputStream)
16:     OutputStream.Flush()
17:     OutputStream.Position = 0
18:
19:     'Return Indented XML String.
20:     PrettyPrintXML = XMLStreamReader.ReadToEnd()
21:     XMLStreamReader.Close()
22: End Function
```

XMLUtilities.TransformXML Method

This method is a general purpose XML transformation method. This method allows arguments to be passed to the XSLT transformation.

C#

```
public String TransformXML();
```

Visual Basic

```
Public Function TransformXML() As String
```

Returns

XML transformed by an XSLT stylesheet.

Remarks

This method takes XML source from

1. A string of xml text.
2. A URI (http://something).
3. A file on the file system.
4. An object of type system.xml.xmldocument.

To take advantage of the first three XML source type options, you must:

a. Set the XMLSourceType (see page 129) property to XMLString, URI, or File.

b. Set the XMLSource (see page 128) property to the string containing the corresponding value.

To take advantage of the fourth XML source type, you must:

a. Set the XMLSourceType (see page 129) property to XMLDocument (see page 128).

b. Set the XMLDocument (see page 128) property to the XML document you wish to transform.

This method takes XSLT source from

1. A string of xml text.
2. A URI (http://something).
3. A file on the file system.

To set the XSLT source type options, you must:

a. Set the XSLTSourceType (see page 129) property to XMLString, URI, or File.

b. Set the XMLSource (see page 128) property to the string containing the corresponding value.

This method takes an arbitrary number of paramaters to pass into the XSLT transformation. In order to pass params into the XSLT transform, you must:

a. Populate a NameValueCollection (param name, param, value).

b. Set the XSLTParams (see page 129) property to your populated NameValueCollection.

Body Source

```
 1: Public Function TransformXML() As String
 2:     Dim myXML As New System.Xml.XmlDocument
 3:     Select Case Me.XMLSourceTypeValue
 4:         Case XMLSourceTypes.File
 5:             Try
 6:                 myXML.Load(System.Web.HttpContext.Current.Server.MapPath(XMLSourceValue))
 7:             Catch ex As Exception
 8:                 Return "Cannot Load XML File: " & Me.XMLSource
 9:             End Try
10:         Case XMLSourceTypes.URI
11:             Try
12:                 myXML.Load(XMLSourceValue)
13:             Catch ex As Exception
14:                 Return "Cannot Load XML File: " & Me.XMLSource
15:             End Try
16:         Case XMLSourceTypes.XMLString
17:             Try
18:                 myXML.LoadXml(XMLSourceValue)
19:             Catch ex As Exception
20:                 Return "Cannot Load XML String: " & ex.Message
21:             End Try
22:         Case XMLSourceTypes.XMLDocument
23:             Try
24:                 myXML = Me.XMLDocument
25:             Catch ex As Exception
26:                 Return "Cannot Load XML Document: " & ex.Message
27:             End Try
28:         Case Else
29:             Try
30:                 Throw New Exception("XML Type Property (File, URI, XMLString, XMLDocument)
Not Set")
```

```
            Catch ex As Exception
                Return "Cannot Load XML: " & ex.Message
            End Try
    End Select

    Dim myXSLT As New System.Xml.Xsl.XslCompiledTransform
    Select Case Me.XSLTSourceTypeValue
        Case XSLTSourceTypes.File
            Try
                myXSLT.Load(System.Web.HttpContext.Current.Server.MapPath(XSLTSourceValue))
            Catch ex As Exception
                Return "Cannot Load XSLT File: " & Me.XSLTSource
            End Try
        Case XSLTSourceTypes.URI
            Try
                myXSLT.Load(XSLTSourceValue)
            Catch ex As Exception
                Return "Cannot Load XSLT File: " & Me.XSLTSource
            End Try
        Case XSLTSourceTypes.XSLTString
            Try
                Dim myXSLTString As New XmlDocument()
                myXSLTString.LoadXml(XSLTSourceValue)
                myXSLT.Load(myXSLTString)
            Catch ex As Exception
                Return "Cannot Load XSLT String: " & ex.Message
            End Try
        Case Else
            Try
                Throw New Exception("XSLT Type Property (File, URI, XSLTString) Not Set")
            Catch ex As Exception
                Return "Cannot Load XSLT: " & ex.Message
            End Try
    End Select

    Dim myOutputStream As New System.IO.MemoryStream()
    Dim myXSLTArgs As New System.Xml.Xsl.XsltArgumentList

    If Not myXSLTParams Is Nothing Then
        Dim i As Integer
        For i = 0 To Me.XSLTParams.Count - 1
            myXSLTArgs.AddParam(Me.XSLTParams.GetKey(i), "", Me.XSLTParams.GetValues(i)(0))
        Next
        Try
            myXSLT.Transform(myXML, myXSLTArgs, myOutputStream)
        Catch ex As Exception
            Return ex.Message
        End Try
    Else
        Try
            myXSLT.Transform(myXML, Nothing, myOutputStream)
        Catch ex As Exception
            Return ex.Message
        End Try
    End If

    myOutputStream.Flush()
    myOutputStream.Position = 0

    Dim myXMLStreamReader As New System.IO.StreamReader(myOutputStream)
    Dim myTransformedXML As String

    myTransformedXML = myXMLStreamReader.ReadToEnd()
    myXMLStreamReader.Close()

    Return myTransformedXML
```

```
97:
98: End Function
```

XMLUtilities Properties

The properties of the XMLUtilities class are listed here.

Public Properties

	Name	Description
	XMLDocument (see page 128)	Holds the XMLDocument to be transformed.
	XMLSource (see page 128)	If XMLSourceType (see page 129) is NOT set to XMLDocument (see page 128), this property holds the location of the XMLDocument (see page 128) to be transformed.
	XMLSourceType (see page 129)	Holds the type of input XML document.
	XSLTParams (see page 129)	Specifies any parameters to be passed into the XSLT transformation.
	XSLTSource (see page 129)	Specifies where the XSLT document is located.
	XSLTSourceType (see page 129)	Sets the type of document to be used as transformation source - specifies where this document lives.

Legend

	Property

XMLUtilities.XMLDocument Property

Holds the XMLDocument to be transformed.

C#

```
public XmlDocument XMLDocument;
```

Visual Basic

```
Public Property XMLDocument() As XmlDocument
```

Returns

XmlDocument

Remarks

Only needs to be populated if the XMLSourceType (see page 129) is set to XmlDocument.

XMLUtilities.XMLSource Property

If XMLSourceType (see page 129) is NOT set to XMLDocument (see page 128), this property holds the location of the XMLDocument (see page 128) to be transformed.

C#

```
public String XMLSource;
```

Visual Basic

```
Public Property XMLSource() As String
```

Returns

The location of the XML Document to be transformed.

Remarks

Only needs to be populated if the XMLSourceType (see page 129) is not set to XMLDocument (see page 128).

XMLUtilities.XMLSourceType Property

Holds the type of input XML document.

C#

```
public XMLSourceTypes XMLSourceType;
```

Visual Basic

```
Public Property XMLSourceType() As XMLSourceTypes
```

Returns

XML Document source type - XMLString, URI, File, or XMLDocument (see page 128).

Remarks

If the type is set to XMLDocument (see page 128), you must set the XMLDocument (see page 128) property to the XMLDocument (see page 128) you wish to transform. Otherwise, set the the XMLSource (see page 128) property to the string location of the XMLDocument (see page 128) you wish to transform.

XMLUtilities.XSLTParams Property

Specifies any parameters to be passed into the XSLT transformation.

C#

```
public NameValueCollection XSLTParams;
```

Visual Basic

```
Public Property XSLTParams() As NameValueCollection
```

Returns

List of parameters to be passed into the transform

Remarks

Optional - can hold any number of parameters. Specified as "param name", "value." You must define these parameters in your XSLT file to use them.

XMLUtilities.XSLTSource Property

Specifies where the XSLT document is located.

C#

```
public String XSLTSource;
```

Visual Basic

```
Public Property XSLTSource() As String
```

Returns

Location of the XSLT document.

XMLUtilities.XSLTSourceType Property

Sets the type of document to be used as transformation source - specifies where this document lives.

C#

```
public XSLTSourceTypes XSLTSourceType;
```

Visual Basic

```
Public Property XSLTSourceType() As XSLTSourceTypes
```

Returns

String

Remarks

Specifies how to load the XSLT document to be used as transformation source.

Ektron.Cms.StarterApps.Wiki Namespace

This is namespace Ektron.Cms.StarterApps.Wiki.

Classes

	Name	Description
	Users (see page 130)	Contains methods for managing users in the Wiki (see page 138) Manager Starter Application.
	Wiki (see page 138)	Contains methods for managing Wiki in the Wiki Manager Starter Application.
	WikiManagement (see page 143)	Contains methods and properties for the overall creation and management of the Wiki (see page 138) application. Important methods worth highlighting are **Initialize (see page 155)** and **GetInformationArchitecture (see page 149).** Initialize (see page 155) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 149) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	XMLUtilities (see page 165)	Contains methods and properties for handling the XML used within the Wiki (see page 138) Starter Application.

Legend

	Class

Classes

The following table lists classes in this documentation.

Classes

	Name	Description
	Users (see page 130)	Contains methods for managing users in the Wiki (see page 138) Manager Starter Application.
	Wiki (see page 138)	Contains methods for managing Wiki in the Wiki Manager Starter Application.
	WikiManagement (see page 143)	Contains methods and properties for the overall creation and management of the Wiki (see page 138) application. Important methods worth highlighting are **Initialize (see page 155)** and **GetInformationArchitecture (see page 149).** Initialize (see page 155) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 149) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	XMLUtilities (see page 165)	Contains methods and properties for handling the XML used within the Wiki (see page 138) Starter Application.

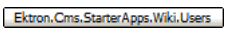
Legend

	Class

Users Class

Contains methods for managing users in the Wiki (see page 138) Manager Starter Application.

Class Hierarchy



C#

```
public class Users;
```

Visual Basic

```
Public Class Users
```

File

Wiki.vb

Users Methods

The methods of the Users class are listed here.

Public Methods

	Name	Description
	AddCMSUserGroup (see page 131)	This method adds a CMS (see page 1) user group to the Wiki (see page 138) Management StarterApp.
	AddMembershipUserGroup (see page 132)	This method adds a Membership user group to the Wiki (see page 138) Management StarterApp.
	AddMembershipUserGroupToFolder (see page 133)	This method adds a Membership user group to a folder and sets the permissions appropriate to the pWiki Management Starer App for the group on the folder.
	AddMemberToWiki (see page 134)	This method adds a user (by user ID) to a user group (by user group ID).
	AddUserGroupToFolder (see page 134)	This method adds a user group to a folder and sets Permissions for the group on the folder.
	AddUserToGroup (see page 136)	This method adds a user (by user ID) to a user group (by group name).
	RemoveMemberFromWiki (see page 136)	This method removes a Membership user from a Wiki (see page 138) user group.
	RemoveUserGroup (see page 137)	This method removes a user group (by user group name).

Legend

	Method

Users.AddCMSUserGroup Method

This method adds a CMS (see page 1) user group to the Wiki (see page 138) Management StarterApp.

C#

```
public int AddCMSUserGroup(String CMSGroupName);
```

Visual Basic

```
Public Function AddCMSUserGroup(ByVal CMSGroupName As String) As Integer
```

Parameters

Parameters	Description
CMSGroupName	The name of the CMS (see page 1) group to add.

Returns

The ID of the newly created CMS (see page 1) user group.

Remarks

This method is meant to be executed only when the StarterApp initializes itself. There is only 1 CMS (see page 1) user group in the StarterApp by design. All members of the CMS (see page 1) user group have permissions to do everything to all folders within the StarterApp.

Body Source

```
1: Public Function AddCMSUserGroup(ByVal CMSGroupName As String) As Integer
2:     Dim User As New Ektron.Cms.API.User.User
3:     Dim UserGroups() As UserGroupData
4:     Dim UserGroup As New UserGroupData
```

```
    Dim UserGroupExists As Boolean = False
    Dim CMSGroupID As Integer

    'See if the CMS user group name already exists.
    UserGroups = User.GetAllUserGroups(EkEnumeration.UserTypes.AuthorType)
    For Each UserGroup In UserGroups
        If UserGroup.GroupName = CMSGroupName Then
            CMSGroupID = UserGroup.GroupId
            UserGroupExists = True
            Exit For
        End If
    Next

    'If the CMS user group doesn't exist, create it.
    If UserGroupExists = False Then
        User.AddUserGroup(CMSGroupName)
        UserGroup = User.GetUserGroupByName(CMSGroupName)
        CMSGroupID = UserGroup.GroupId
    End If

    'return the CMS user group ID.
    Return CMSGroupID
End Function
```

Users.AddMembershipUserGroup Method

This method adds a Membership user group to the Wiki (see page 138) Management StarterApp.

C#

```
public int AddMembershipUserGroup(String MemershipGroupName);
```

Visual Basic

```
Public Function AddMembershipUserGroup(ByVal MemershipGroupName As String) As Integer
```

Parameters

Parameters	Description
MemershipGroupName	The name of the Membership user group to add.

Returns

The ID of the newly created Membership user group.

Remarks

This method is executed once per Wiki (see page 138). That is, each Wiki (see page 138) gets its own Membership user group. The Wiki (see page 138)'s Membership user group is deleted when the Wiki (see page 138) is removed with the RemoveWiki() method.

Body Source

```
Public Function AddMembershipUserGroup(ByVal MemershipGroupName As String) As Integer
    Dim MembershipUserGroupCollection As New Collection
    Dim User As New Ektron.Cms.API.User.User
    Dim UserGroups() As UserGroupData
    Dim UserGroup As New UserGroupData
    Dim UserGroupExists As Boolean = False
    Dim MembershipGroupID As Integer
    Dim ContentAPI As New Ektron.Cms.ContentAPI
    Dim ErrorMessage As String = ""

    'See if the CMS user group name already exists.
    UserGroups = User.GetAllUserGroups(EkEnumeration.UserTypes.AuthorType)
    For Each UserGroup In UserGroups
        If UserGroup.GroupName = MemershipGroupName Then
            MembershipGroupID = UserGroup.GroupId
```

```
16:             UserGroupExists = True
17:             Exit For
18:         End If
19:     Next
20:
21:     'If user group doesn't exist, create it.
22:     If UserGroupExists = False Then
23:         MembershipUserGroupCollection.Add(MemershipGroupName, "UserGroupName")
24:         MembershipUserGroupCollection.Add("StarterApp membership usergroup", "Description")
25:         Try
26:             Dim tmpID As Integer = ContentAPI.RequestInformationRef.CallerId
27:             ContentAPI.RequestInformationRef.CallerId = EkConstants.InternalAdmin
28:             ContentAPI.EkUserRef.AddMemberShipGroupV4(MembershipUserGroupCollection, "",
"")
29:             ContentAPI.RequestInformationRef.CallerId = tmpID
30:
31:             UserGroup = User.GetUserGroupByName(MemershipGroupName)
32:             MembershipGroupID = UserGroup.GroupId
33:         Catch ex As Exception
34:             MembershipGroupID = -1
35:             ErrorMessage = ErrorMessage & "Membership Group Could Not Be Created.  "
36:             ErrorMessage = ErrorMessage & "You may manually work around this error by
creating a Membership usergroup in the Workarea.  "
37:             ErrorMessage = ErrorMessage & "The Membership usergroup name must be '" &
MemershipGroupName & "'."
38:             Throw New Exception(ErrorMessage & ex.Message)
39:         End Try
40:
41:     End If
42:
43:     'return the user group ID.
44:     Return MembershipGroupID
45: End Function
```

Users.AddMembershipUserGroupToFolder Method

This method adds a Membership user group to a folder and sets the permissions appropriate to the pWiki Management Starer App for the group on the folder.

C#

```
public AddMembershipUserGroupToFolder(int FolderID, int GroupID);
```

Visual Basic

```
Public Sub AddMembershipUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer)
```

Parameters

Parameters	Description
FolderID	The folder to which you wish to assign the user group.
GroupID	The group ID of the Membership user group you wish to assign to the folder.

Remarks

This method includes the permission settings appropriate to Membership users (see page 130) in the pWiki Management Starter App.

Body Source

```
 1: Public Sub AddMembershipUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As
Integer)
 2:     Dim UserPermissionData As New Ektron.Cms.UserPermissionData
 3:     Dim Permissions As New Ektron.Cms.API.Permissions
 4:
 5:     'Add permissions for GroupID to FolderID.
 6:     UserPermissionData.GroupId = GroupID
 7:     UserPermissionData.FolderId = FolderID
```

```
 8:     'File/folder permissions.
 9:     UserPermissionData.CanAdd = True
10:     UserPermissionData.CanEdit = True
11:     UserPermissionData.CanDelete = False
12:     'Library permissions.
13:     UserPermissionData.CanAddToFileLib = True
14:     UserPermissionData.CanAddToImageLib = True
15:     UserPermissionData.CanAddToHyperlinkLib = False
16:     UserPermissionData.CanOverwriteLib = False
17:     'Membership users can traverse folders by default (hardcoded to True).
18:     'We are "adding" here to show explicitly that this permission setting is required.
19:     UserPermissionData.CanTraverseFolders = True
20:     Try
21:         Permissions.AddItemPermission(UserPermissionData)
22:     Catch ex As Exception
23:
24:     End Try
25:
26: End Sub
```

Users.AddMemberToWiki Method

This method adds a user (by user ID) to a user group (by user group ID).

C#

```
public AddMemberToWiki();
```

Visual Basic

```
Public Sub AddMemberToWiki()
```

Remarks

This method requires population of two Users (see page 130) properties...

1. MemberID (see page 138)
2. WikiID

This method is more or less the same thing as AddUserToGroup (see page 136)() only this method takes a user group ID whereas AddUserToGroup (see page 136)() takes a user group name.

Body Source

```
 1: Public Sub AddMemberToWiki()
 2:     Dim Member As New Ektron.Cms.API.User.User
 3:     Dim GroupInfo As New UserGroupData
 4:     Dim Members As New Ektron.Cms.API.User.User
 5:     Dim FolderAPI As New Ektron.Cms.API.Folder
 6:
 7:     GroupInfo = Member.GetUserGroupByName("wiki." & FolderAPI.GetFolder(Me.wikiID).Name)
 8:     Try
 9:         Members.AddUserToGroup(Me.MemberID, GroupInfo.GroupId)
10:     Catch ex As Exception
11:         'Do Nothing.
12:     End Try
13: End Sub
```

Users.AddUserGroupToFolder Method

This method adds a user group to a folder and sets Permissions for the group on the folder.

C#

```
public AddUserGroupToFolder(int FolderID, int GroupID, Boolean TraverseOnly);
```

Visual Basic

```
Public Sub AddUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer, ByVal
```

```
TraverseOnly As Boolean)
```

Parameters

Parameters	Description
FolderID	The folder to which to assign the user group.
GroupID	The group ID of the user group to assign to the folder.
TraverseOnly	**True** - the group will be given Traverse permissions only. **False** - the group is given all permissions.

Remarks

This method can be used as a general-purpose method. It will work against any folder not just the ones in the StarterApp. Traverse-only is meaningful only to CMS (see page 1) user groups. Membership user groups always have traverse permissions set to True (this is hard coded in the CMS (see page 1)).

Body Source

```
 1: Public Sub AddUserGroupToFolder(ByVal FolderID As Integer, ByVal GroupID As Integer, ByVal
TraverseOnly As Boolean)
 2:     Dim UserPermissionData As New Ektron.Cms.UserPermissionData
 3:     Dim Permissions As New Ektron.Cms.API.Permissions
 4:
 5:     'Add permissions for GroupID to FolderID.
 6:     UserPermissionData.GroupId = GroupID
 7:     UserPermissionData.FolderId = FolderID
 8:
 9:     If TraverseOnly = False Then
10:         'Grant everything.
11:         UserPermissionData.CanAddToFileLib = True
12:         UserPermissionData.CanOverwriteLib = True
13:         UserPermissionData.CanAddToHyperlinkLib = True
14:         UserPermissionData.CanAddToImageLib = True
15:         UserPermissionData.CanAddToQuicklinkLib = True
16:         UserPermissionData.CanApprove = True
17:         UserPermissionData.CanBreakPending = True
18:         UserPermissionData.CanCreateTask = True
19:         UserPermissionData.CanDecline = True
20:         UserPermissionData.CanAdd = True
21:         UserPermissionData.CanDelete = True
22:         UserPermissionData.CanEdit = True
23:         UserPermissionData.CanHistory = True
24:         UserPermissionData.CanPreview = True
25:         UserPermissionData.CanPublish = True
26:         UserPermissionData.CanRestore = True
27:         UserPermissionData.CanView = True
28:         UserPermissionData.CanDeleteFolders = True
29:         UserPermissionData.CanAddFolders = True
30:         UserPermissionData.CanEditFolders = True
31:         UserPermissionData.CanEditCollections = True
32:         UserPermissionData.CanTraverseFolders = True
33:     Else
34:         'Grant only Traverse and Read-Only.
35:         UserPermissionData.IsReadOnly = True
36:         UserPermissionData.IsReadOnlyLib = False
37:         UserPermissionData.CanAddToFileLib = False
38:         UserPermissionData.CanOverwriteLib = False
39:         UserPermissionData.CanAddToHyperlinkLib = False
40:         UserPermissionData.CanAddToImageLib = False
41:         UserPermissionData.CanAddToQuicklinkLib = False
42:         UserPermissionData.CanApprove = False
43:         UserPermissionData.CanBreakPending = False
44:         UserPermissionData.CanCreateTask = False
45:         UserPermissionData.CanDecline = False
46:         UserPermissionData.CanAdd = False
47:         UserPermissionData.CanDelete = False
```

```
48:         UserPermissionData.CanEdit = False
49:         UserPermissionData.CanHistory = False
50:         UserPermissionData.CanPreview = False
51:         UserPermissionData.CanPublish = False
52:         UserPermissionData.CanRestore = False
53:         UserPermissionData.CanView = False
54:         UserPermissionData.CanDeleteFolders = False
55:         UserPermissionData.CanAddFolders = False
56:         UserPermissionData.CanEditFolders = False
57:         UserPermissionData.CanEditCollections = False
58:         UserPermissionData.CanTraverseFolders = True
59:     End If
60:
61:
62:     Try
63:         Permissions.AddItemPermission(UserPermissionData)
64:     Catch ex As Exception
65:
66:     End Try
67:
68: End Sub
```

Users.AddUserToGroup Method

This method adds a user (by user ID) to a user group (by group name).

C#

```
public AddUserToGroup(int UserID, String GroupName);
```

Visual Basic

```
Public Sub AddUserToGroup(ByVal UserID As Integer, ByVal GroupName As String)
```

Parameters

Parameters	Description
UserID	The ID of the user to add to the group.
GroupName	The name of the group to which to add the user.

Remarks

This method is more or less the same thing as AddMembertoWiki (see page 134)() only this method takes a user group name whereas AddMemberToWiki (see page 134)() takes a user group ID.

Body Source

```
 1: Public Sub AddUserToGroup(ByVal UserID As Integer, ByVal GroupName As String)
 2:     Dim UserGroup As New Ektron.Cms.API.User.User
 3:     Dim GroupInfo As New UserGroupData
 4:     Dim Users As New Ektron.Cms.API.User.User
 5:
 6:     GroupInfo = UserGroup.GetUserGroupByName(GroupName)
 7:     If GroupInfo IsNot Nothing Then
 8:         Try
 9:             Users.AddUserToGroup(UserID, GroupInfo.GroupId)
10:         Catch ex As Exception
11:
12:         End Try
13:     End If
14: End Sub
```

Users.RemoveMemberFromWiki Method

This method removes a Membership user from a Wiki (see page 138) user group.

C#

```
public RemoveMemberFromWiki();
```

Visual Basic

```
Public Sub RemoveMemberFromWiki()
```

Remarks

This method requires population of two Users (see page 130) properties...

1. MemberID (see page 138)
2. WikiID

Body Source

```
 1: Public Sub RemoveMemberFromWiki()
 2:     Dim Member As New Ektron.Cms.API.User.User
 3:     Dim GroupInfo As New UserGroupData
 4:     Dim FolderAPI As New Ektron.Cms.API.Folder
 5:
 6:     GroupInfo = Member.GetUserGroupByName("wiki." & FolderAPI.GetFolder(Me.wikiID).Name)
 7:     Try
 8:         Member.DeleteUserFromGroup(Me.MemberID, GroupInfo.GroupId, False)
 9:     Catch ex As Exception
10:         'Do Nothing.
11:     End Try
12: End Sub
```

Users.RemoveUserGroup Method

This method removes a user group (by user group name).

C#

```
public RemoveUserGroup(String GroupName);
```

Visual Basic

```
Public Sub RemoveUserGroup(ByVal GroupName As String)
```

Parameters

Parameters	Description
GroupName	The name of the user group to remove.

Remarks

This method is called by RemoveWiki() method. When a Wiki (see page 138)/wiki (see page 130) is removed, so is its Membership user group.

Body Source

```
 1: Public Sub RemoveUserGroup(ByVal GroupName As String)
 2:     Dim UserGroup As New API.User.User
 3:     Dim UserGroupData As New UserGroupData
 4:     Try
 5:         UserGroupData =
UserGroup.GetUserGroupByName(Ektron.Cms.Common.EkFunctions.HtmlDecode(GroupName))
 6:         UserGroup.DeleteUserGroup(UserGroupData.GroupId)
 7:     Catch ex As Exception
 8:
 9:     End Try
10: End Sub
```

Users Properties

The properties of the Users class are listed here.

Public Properties

	Name	Description
	MemberID (see page 138)	The MemberID property is used to hold the ID of a membership user created at run-time via Registration widgets.
	wikiID (see page 138)	Holds the ID of the selected Wiki (see page 138).

Legend

	Property

Users.MemberID Property

The MemberID property is used to hold the ID of a membership user created at run-time via Registration widgets.

C#

```
public int MemberID;
```

Visual Basic

```
Public Property MemberID() As Integer
```

Returns

The ID of the membership user created at run-time.

Users.wikiID Property

Holds the ID of the selected Wiki (see page 138).

C#

```
public int wikiID;
```

Visual Basic

```
Public Property wikiID() As Integer
```

Returns

The selected Wiki (see page 138)'s ID.

Wiki Class

Contains methods for managing Wiki in the Wiki Manager Starter Application.

Class Hierarchy

C#

```
public class Wiki;
```

Visual Basic

```
Public Class Wiki
```

File

Wiki.vb

Wiki Methods

The methods of the Wiki class are listed here.

Private Methods

	Name	Description
	DefaultBlogContentText (see page 139)	Returns default Blog content.
	LoremIpsumLong (see page 141)	This method generates placeholder text.
	LoremIpsumShort (see page 142)	This method generates placeholder text.

Legend

	Method
	Private

Public Methods

	Name	Description
	GetAllWiki (see page 139)	This method gathers all the Wiki (see page 138) names and Wiki (see page 138) folder ID's that the current user has permissions to view.
	GetWikiXMLForInformationArchitecture (see page 140)	This method gathers the Information Architecture for a particular Wiki (see page 138).
	RemoveWiki (see page 142)	This method removes a Wiki (see page 138) from the Wiki (see page 138) StarterApp.

Legend

	Method

Wiki.DefaultBlogContentText Method

Returns default Blog content.

C#

```
private String DefaultBlogContentText();
```

Visual Basic

```
Private Function DefaultBlogContentText() As String
```

Returns

String

Remarks

This method is used to insert content into a content block placeholder when creating a new Blog.

Body Source

```
1: Private Function DefaultBlogContentText() As String
2:     Dim text As StringBuilder = New StringBuilder()
3:     text.Append("<h3>Highlights:</h3><p> </p>")
4:     text.Append("<h3>Status:</h3><p> </p>")
5:     text.Append("<h3>Fires:</h3><p> </p>")
6:
7:     Return text.ToString()
8: End Function
```

Wiki.GetAllWiki Method

This method gathers all the Wiki (see page 138) names and Wiki (see page 138) folder ID's that the current user has permissions to view.

C#

```
public NameValueCollection GetAllWiki();
```

Visual Basic

```
Public Function GetAllWiki() As NameValueCollection
```

Returns

NameValueCollection - Wiki (see page 138) name, Wiki (see page 138) folder ID.

Remarks

This method uses GetChildFolders and therefore respects permissions.

Body Source

```
 1: Public Function GetAllWiki() As NameValueCollection
 2:     Dim WM As New Ektron.Cms.StarterApps.Wiki.WikiManagement
 3:     Dim WMFolderID As Integer
 4:     Dim WMFolderData() As FolderData
 5:     Dim WMFolderItem As New FolderData
 6:     Dim Wiki As New NameValueCollection
 7:     Dim FolderAPI As New Ektron.Cms.API.Folder
 8:
 9:     WMFolderID = WM.Initialize()
10:
11:     'Create nameValueCollection of folder names and folder IDs.
12:     WMFolderData = FolderAPI.GetChildFolders(WMFolderID, False)
13:     If Not WMFolderData Is Nothing Then
14:         For Each WMFolderItem In WMFolderData
15:             Wiki.Add(WMFolderItem.Name, WMFolderItem.Id)
16:         Next
17:     End If
18:
19:     Return Wiki
20: End Function
```

Wiki.GetWikiXMLForInformationArchitecture Method

This method gathers the Information Architecture for a particular Wiki (see page 138).

C#

```
public System.Xml.XmlDocument GetWikiXMLForInformationArchitecture(int wikiID);
```

Visual Basic

```
Public Function GetWikiXMLForInformationArchitecture(ByVal wikiID As Integer) As
System.Xml.XmlDocument
```

Parameters

Parameters	Description
wikiID	The ID of the Wiki (see page 138) (folder ID) for which the IA is to be generated.

Returns

An XML document containing the Information Architecture for a particular Wiki (see page 138).

Remarks

This method is a more efficient version of Wiki.GetInformationArchitectureXML(), which returns the entire IA tree for a particular user. This method restricts the tree to a particular branch (one Wiki (see page 138)), so it's a bit more efficient. pWiki.GetInformationArchitect() also has a boolean switch to include all the members of the Wiki (see page 138)'s Membership user group. This method does not have such a switch.

Body Source

```
 1: Public Function GetWikiXMLForInformationArchitecture(ByVal wikiID As Integer) As
System.Xml.XmlDocument
 2:
```

```
 3:     Dim WM As New Ektron.Cms.StarterApps.Wiki.WikiManagement
 4:     Dim DocumentNode, WikiNode As System.Xml.XmlNode
 5:     Dim XMLUtils As New Ektron.Cms.StarterApps.Wiki.XMLUtilities
 6:     Dim FolderAPI As New Ektron.Cms.API.Folder
 7:     Dim WikiFolderData As New Ektron.Cms.FolderData
 8:     Dim FolderData As New Ektron.Cms.FolderData
 9:
10:     GetWikiXMLForInformationArchitecture = New System.Xml.XmlDocument
11:
12:     FolderData = FolderAPI.GetFolder(wikiID)
13:
14:     'Initalize the IA XML document.
15:     DocumentNode = GetWikiXMLForInformationArchitecture.CreateElement("ektron")
16:     GetWikiXMLForInformationArchitecture.AppendChild(DocumentNode)
17:
18:     WikiNode = GetWikiXMLForInformationArchitecture.CreateElement("li")
19:     XMLUtils.ApplyXMLAttribute(GetWikiXMLForInformationArchitecture, WikiNode, "id",
wikiID)
20:     XMLUtils.ApplyXMLAttribute(GetWikiXMLForInformationArchitecture, WikiNode, "label",
FolderData.Name)
21:     XMLUtils.ApplyXMLAttribute(GetWikiXMLForInformationArchitecture, WikiNode, "class",
"Wiki")
22:     GetWikiXMLForInformationArchitecture.DocumentElement.AppendChild(WikiNode)
23:
24:     Return GetWikiXMLForInformationArchitecture
25: End Function
```

Wiki.LoremIpsumLong Method

This method generates placeholder text.

C#

```
private String LoremIpsumLong();
```

Visual Basic

```
Private Function LoremIpsumLong() As String
```

Returns

3 html paragraphs of Lorem Ipsum text.

Remarks

This is here to save the hassle of cutting and pasting placeholder text.

Body Source

```
1: Private Function LoremIpsumLong() As String
2:     LoremIpsumLong = "<p>" & "Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Donec pharetra, nisl tristique semper rutrum, risus magna congue metus, faucibus venenatis
orci metus vel lectus. Duis at ante. Donec bibendum. In lectus orci, rhoncus hendrerit, cursus
dictum, auctor pharetra, tortor. Maecenas metus mi, mollis non, iaculis eget, ornare in,
risus. Cras ut odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere
cubilia Curae; Vestibulum eu lorem. Suspendisse quis mauris. Quisque nisl. Maecenas felis.
Fusce dui odio, tristique sit amet, tincidunt et, volutpat a, diam. Phasellus non libero.
Phasellus et leo a libero cursus congue. Pellentesque feugiat est nec nulla." & "</p>"
3:     LoremIpsumLong = LoremIpsumLong & "<p>" & "Proin et enim interdum nisl elementum
porttitor. Etiam quam turpis, hendrerit quis, dictum id, ultricies quis, tortor. Vestibulum
ante metus, aliquet quis, posuere at, pharetra at, lorem. Etiam nec felis. Integer fringilla
bibendum arcu. Pellentesque iaculis libero vitae libero. Aenean euismod mollis lorem. Etiam
sit amet arcu. Nunc nisi dolor, luctus vel, rhoncus commodo, tempor ut, turpis. Praesent
auctor vehicula erat. Sed porttitor accumsan massa. Phasellus sed lectus." & "</p>"
4:     LoremIpsumLong = LoremIpsumLong & "<p>" & "Nunc porta ornare pede. Donec tincidunt
elementum mauris. Aliquam interdum elit id neque. Donec quis eros ac tellus nonummy laoreet.
Nam ultrices, risus sit amet molestie dignissim, dolor turpis tempus est, in semper dui risus
vel nulla. Morbi iaculis metus at ipsum. Praesent lectus pede, malesuada eget, bibendum quis,
bibendum luctus, orci. Pellentesque rhoncus lacus a mi. Nam vulputate lorem eget neque.
```

```
Aliquam aliquet purus in erat. Vivamus dignissim fringilla enim. Curabitur id magna. Maecenas
aliquet posuere ante. Proin arcu tortor, cursus in, facilisis quis, sagittis sit amet, est.
Nunc et erat at ipsum semper ultricies. Sed mi. Vestibulum sed nisi. Donec massa nisl, posuere
sed, pharetra et, egestas vitae, magna." & "</p>"
5:     Return LoremIpsumLong
6: End Function
```

Wiki.LoremIpsumShort Method

This method generates placeholder text.

C#

```
private String LoremIpsumShort();
```

Visual Basic

```
Private Function LoremIpsumShort() As String
```

Returns

1 html paragraphs of Lorem Ipsum text.

Remarks

This is here to save the hassle of cutting and pasting placeholder text.

Body Source

```
1: Private Function LoremIpsumShort() As String
2:     LoremIpsumShort = "<p>" & "Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
Donec pharetra, nisl tristique semper rutrum, risus magna congue metus, faucibus venenatis
orci metus vel lectus. Duis at ante. Donec bibendum. In lectus orci, rhoncus hendrerit, cursus
dictum, auctor pharetra, tortor. Maecenas metus mi, mollis non, iaculis eget, ornare in,
risus. Cras ut odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere
cubilia Curae; Vestibulum eu lorem. Suspendisse quis mauris. Quisque nisl. Maecenas felis.
Fusce dui odio, tristique sit amet, tincidunt et, volutpat a, diam. Phasellus non libero.
Phasellus et leo a libero cursus congue. Pellentesque feugiat est nec nulla." & "</p>"
3:     Return LoremIpsumShort
4: End Function
```

Wiki.RemoveWiki Method

This method removes a Wiki (see page 138) from the Wiki (see page 138) StarterApp.

C#

```
public RemoveWiki(int WikiFolderID);
```

Visual Basic

```
Public Sub RemoveWiki(ByVal WikiFolderID As Integer)
```

Parameters

Parameters	Description
WikiFolderID	The ID of the wiki (see page 138) (folder ID) to be removed.

Remarks

1. This method removes the folder taxonomy
2. This method removes the groups, "wiki.mem.~Wiki (see page 138) name~." and "wiki (see page 138).~Wikiname"
3. This method removes the wiki (see page 138) folder.

Body Source

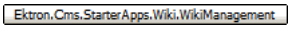
```
1: Public Sub RemoveWiki(ByVal WikiFolderID As Integer)
2:     Dim UserGroup As New Ektron.Cms.StarterApps.Wiki.Users
3:     Dim FolderAPI As New Ektron.Cms.API.Folder
4:     Dim Taxonomy As New Ektron.Cms.API.Content.Taxonomy
```

```
 5:     Dim TaxonomyBaseData() As Ektron.Cms.TaxonomyBaseData
 6:     Dim TaxonomyBaseDatum As New Ektron.Cms.TaxonomyBaseData
 7:     Dim TaxonomyRequest As New Ektron.Cms.TaxonomyRequest
 8:     Dim r_WikiName As String
 9:     Dim FolderData As New Ektron.Cms.FolderData
10:
11:
12:     FolderData = FolderAPI.GetFolder(WikiFolderID, True)
13:     r_WikiName = FolderData.Name
14:     TaxonomyBaseData = FolderData.FolderTaxonomy
15:
16:     'Delete the wiki's taxonomy.
17:     For Each TaxonomyBaseDatum In TaxonomyBaseData
18:         If TaxonomyBaseDatum.TaxonomyName = "wiki." &
Ektron.Cms.Common.EkFunctions.HtmlDecode(r_WikiName) Then
19:
20:             TaxonomyRequest.TaxonomyId = TaxonomyBaseDatum.TaxonomyId
21:             TaxonomyRequest.TaxonomyLanguage = FolderAPI.DefaultContentLanguage
22:             Taxonomy.DeleteTaxonomy(TaxonomyRequest)
23:             Exit For
24:         End If
25:     Next
26:
27:     'Delete wiki Membership user group.
28:     Try
29:         UserGroup.RemoveUserGroup("wiki." & FolderAPI.GetFolder(WikiFolderID).Name)
30:         UserGroup.RemoveUserGroup("wiki.mem." & FolderAPI.GetFolder(WikiFolderID).Name)
31:     Catch ex As Exception
32:
33:     End Try
34:
35:     'Delete wiki folder.
36:     Try
37:         FolderAPI.DeleteFolderById(WikiFolderID)
38:     Catch ex As Exception
39:
40:     End Try
41:
42: End Sub
```

WikiManagement Class

Contains methods and properties for the overall creation and management of the Wiki (see page 138) application. Important methods worth highlighting are **Initialize (see page 155)** and **GetInformationArchitecture (see page 149).** Initialize (see page 155) is executed upon each page load to ensure the Starter Application has all the components it needs. GetInformationArchitecture (see page 149) gets the information architecture (navigation/content hierarchy) the current user has permissions to access.

Class Hierarchy



C#

```
public class WikiManagement;
```

Visual Basic

```
Public Class WikiManagement
```

File

Wiki.vb

WikiManagement Enumerations

The enumerations of the WikiManagement class are listed here.

Enumerations

	Name	Description
	InformationArchitectureUserModes (see page 144)	**MembershipUser** - can add content to Wiki (see page 138). They cannot remove members,content or Wiki (see page 138) **CMSUser** - can add members,content and Wiki (see page 138). In addition, a CMSUser can remove members,content or Wiki (see page 138).
	MetadataDefinitionTypes (see page 144)	This enum holds the text for the three metdata definition types used by the Wiki (see page 138) Management StarterApp.

Legend

	Enumeration

Ektron.Cms.StarterApps.Wiki.WikiManagement.InformationArchitectureUserModes Enumeration

MembershipUser - can add content to Wiki (see page 138). They cannot remove members,content or Wiki (see page 138)

CMSUser - can add members,content and Wiki (see page 138). In addition, a CMSUser can remove members,content or Wiki (see page 138).

C#

```
public enum InformationArchitectureUserModes {
  MembershipUser,
  CMSUser
}
```

Visual Basic

```
Public Enum InformationArchitectureUserModes
  MembershipUser,
  CMSUser
End Enum
```

File

Wiki.vb

Members

Members	Description
MembershipUser	Can add members and content to Wiki (see page 138). Cannot remove members or Wiki (see page 138).
CMSUser	Can add members,content and Wiki (see page 138). In addition, a CMSUser can remove members,content and Wiki (see page 138).

Remarks

This enum is used by the Public Property InformationArchitectureUserMode (see page 163)().

Ektron.Cms.StarterApps.Wiki.WikiManagement.MetadataDefinitionTypes Enumeration

This enum holds the text for the three metdata definition types used by the Wiki (see page 138) Management StarterApp.

C#

```
public enum MetadataDefinitionTypes {
}
```

Visual Basic

```
Public Enum MetadataDefinitionTypes
End Enum
```

File

Wiki.vb

Remarks

These metadata definition types are added in Initialize (see page 155)() and consumed during generation of InformationArchitecture.xml.

WikiManagement Fields

The fields of the WikiManagement class are listed here.

Private Fields

	Name	Description
	InformationAchitectureUserModeValues (see page 145)	For use with Public Property Set InformationArchitectureUserMode (see page 163)() Set "MembershipUser" as default - this is a precaution; MembershipUser cannot delete Wiki (see page 138) and cannot add Wiki (see page 138).

Legend

	Data Member
	Private

WikiManagement.InformationAchitectureUserModeValues Field

C#

```
private InformationArchitectureUserModes InformationAchitectureUserModeValues =
InformationArchitectureUserModes.MembershipUser;
```

Visual Basic

```
Private InformationAchitectureUserModeValues As InformationArchitectureUserModes =
InformationArchitectureUserModes.MembershipUser
```

Description

For use with Public Property Set InformationArchitectureUserMode (see page 163)() Set "MembershipUser" as default - this is a precaution; MembershipUser cannot delete Wiki (see page 138) and cannot add Wiki (see page 138).

WikiManagement Methods

The methods of the WikiManagement class are listed here.

Private Methods

	Name	Description
	AddFolder (see page 146)	Creates StarterApps (see page 1) and Wiki (see page 138) Management folders.
	AddMetadataDefinition (see page 147)	This method is called by Initialize (see page 155)() when creating the required metadata values
	GetBlogPermissionsType (see page 148)	This method gets the logical "type" of blog.
	GetMembersXML (see page 151)	This method inserts member-information to the Wiki (see page 138) nodes in the InformationArchitecture.xml generated by GetInformationArchitectureXML (see page 150)().
	GetwikiXML (see page 154)	This method collects blog component information and inserts this information into InformationArchitecture.xml. This method recursively calls itself to gather all folders/sub-folders of a particular Wiki (see page 138).
	MembershipUserTraverseOnly (see page 158)	This method checks to see if a Membership user ID has been specifically granted permissions to a folder.
	RegisterTemplate (see page 159)	This method registers our templates inside CMS400 during Wiki (see page 138) Management StarterApp Initialization.
	RemoveInheritedUserGroups (see page 160)	This method removes any user groups and users assigned to a folder.

Legend

	Method
	Private

Public Methods

	Name	Description
	GetBreadcrumbs (see page 149)	This method gets the breadcrumb trail based on the user's navigation selection.
	GetInformationArchitecture (see page 149)	This method gets the information architecture (navigation/content hierarchy) the current user has permissions to access.
	GetInformationArchitectureXML (see page 150)	This method generates an XML-based hierarchy of Wiki (see page 138) and its components to which the user has permissions.
	GetMetadataDefinitionID (see page 153)	This method gets IDs for any of the three metadata definitions for the Wiki (see page 138) Management StarterApp.
	Initialize (see page 155)	The Initialize() method is executed upon each page load to ensure StartApp has the components it needs, and returns the "Wiki (see page 138) Management" folder ID.
	SetFolderPermissions (see page 161)	This method breaks content and metadata inheritance for folders. You may specify which inheritance property you wish to break or specify if you wish to break both.
	SetFolderPrivate (see page 161)	This method sets the folder to private status.
	TransformInformationArchitectureXML (see page 162)	This method transforms Information Architecture XML to an XHTML unordered list.

Legend

	Method

WikiManagement.AddFolder Method

Creates StarterApps (see page 1) and Wiki (see page 138) Management folders.

C#

```
private int AddFolder(String FolderName, int ParentID);
```

Visual Basic

```
Private Function AddFolder(ByVal FolderName As String, ByVal ParentID As Integer) As Integer
```

Parameters

Parameters	Description
FolderName	This should either be "StarterApps (see page 1)" or "Wiki (see page 138) Management."
ParentID	This should either by "0" if it's the "StarterApps (see page 1)" folder or StarterAppsFolderID if it's the "Wiki (see page 138) Management" folder.

Returns

Folder ID

Remarks

This method is called by the Intialize() method.

Body Source

```
 1: Private Function AddFolder(ByVal FolderName As String, ByVal ParentID As Integer) As
Integer
 2:
 3:     Dim FolderRequest As New FolderRequest
 4:     Dim FolderDataItem As New FolderData
 5:
 6:     FolderRequest.FolderName = FolderName
 7:     FolderRequest.ParentId = ParentID
 8:     FolderRequest.MetaInherited = 0
 9:     FolderRequest.StyleSheet = "NoCss.css"
```

```
10:     FolderRequest.TemplateFileName = "NoTemplate.aspx"
11:     FolderRequest.TaxonomyInherited = False
12:     FolderRequest.CategoryRequired = False
13:     FolderRequest.FolderType = Ektron.Cms.Common.EkEnumeration.FolderType.Content
14:     FolderRequest.suppressNotification = True
15:     FolderRequest.FolderDescription = "Ektron Starter Apps Folder"
16:     FolderRequest.SiteMapPathInherit = False
17:     FolderAPI.AddFolder(FolderRequest)
18:
19:     Return FolderRequest.FolderId
20: End Function
```

WikiManagement.AddMetadataDefinition Method

This method is called by Initialize (see page 155)() when creating the required metadata values

C#

```
private int AddMetadataDefinition(MetadataDefinitionTypes Type);
```

Visual Basic

```
Private Function AddMetadataDefinition(ByVal Type As MetadataDefinitionTypes) As Integer
```

Parameters

Parameters	Description
Type	Values: "WM.DefaultContent", "WM.MembershipBlog", "WM.CMSBlog"

Returns

Metadata definition ID

Remarks

- **"wiki.DefaultContent"** - used to identify a content entry to load by default when the Wiki (see page 138) loads without any user-selected content.
- **"wiki.MembershipBlog"** - used during the creation of InformationArchitecture.xml to identify the audience of the blog ("Membership Blogs" are actually viewable by both CMS (see page 1) users AND Membership users.)
- **"wiki.CMSBlog"** - used during the creation of InformationArchitecture.xml to identify the audience of the blog ("CMS (see page 1) Blogs" are viewable only by CMS (see page 1) users.)

Body Source

```
 1: Private Function AddMetadataDefinition(ByVal Type As MetadataDefinitionTypes) As Integer
 2:     Dim Metadata As New Metadata
 3:     Dim MetadataCollection As New Ektron.Cms.ContentMetaData
 4:     Dim MetadataTypeName As String
 5:
 6:     Select Case Type
 7:         Case MetadataDefinitionTypes.Wiki
 8:             MetadataTypeName = "wiki.DefaultContent"
 9:         Case MetadataDefinitionTypes.MembershipBlog
10:             MetadataTypeName = "wiki.MembershipBlog"
11:         Case MetadataDefinitionTypes.CMSBlog
12:             MetadataTypeName = "wiki.CMSBlog"
13:         Case Else
14:             Exit Function
15:     End Select
16:
17:     MetadataCollection.TypeName = MetadataTypeName
18:     MetadataCollection.DefaultText = "No"
19:
20:     'Title Values:
21:     'text, number, byte, double, float, integer, long, short, date, boolean, select1
(select from a list), select (multiple selections).
22:     MetadataCollection.Title = "boolean"
```

```
    'TagType Values:
    '100 - Searchable Property
    '1 - Meta
    '0 - HTML
    '2 - Collection
    '3 - List Summary
    '4 - Content
    '5 - Image
    '7 - File
    MetadataCollection.TagType = "100"
    MetadataCollection.Editable = True
    MetadataCollection.Separater = ";"
    MetadataCollection.SearchAllowed = True
    MetadataCollection.SelectableText = "No;Yes"
    MetadataCollection.MetaDisplayEE = False
    MetadataCollection.MetaDisplay = False
    MetadataCollection.IsSelectableOnly = True

    Try
        Metadata.AddMetaDataType(MetadataCollection)
    Catch ex As Exception
        Return -1
    End Try

    Return MetadataCollection.TypeId

End Function
```

WikiManagement.GetBlogPermissionsType Method

This method gets the logical "type" of blog.

C#

```
private String GetBlogPermissionsType(int BlogID);
```

Visual Basic

```
Private Function GetBlogPermissionsType(ByVal BlogID As Integer) As String
```

Parameters

Parameters	Description
BlogID	ID of a blog inside the Wiki (see page 138) Management StarterApp folder tree.

Returns

String: either "MembershipBlog", or "CMSBlog."

Remarks

This value is based on a metadata setting assigned at the time the blog was created. This value differentiates blogs intended for CMS (see page 1) users from those intended for CMS (see page 1) AND Membership users.

Body Source

```
Private Function GetBlogPermissionsType(ByVal BlogID As Integer) As String
    Dim Metadata As New Ektron.Cms.API.CustomFields
    Dim MetadataCollection As New Collection
    Dim MetadataType As String = "None"
    Dim BlogTypeArray(0) As String
    Dim BlogType As String

    MetadataCollection = Metadata.GetFieldsByFolder(BlogID, 1033)
    For Each MetadataItem As Collection In MetadataCollection
        If MetadataItem IsNot Nothing Then
            If MetadataItem.Item("Assigned") <> 0 Then
                MetadataType = MetadataItem.Item("CustomFieldName")
            End If
```

```
14:         End If
15:     Next
16:     BlogTypeArray = Split(MetadataType, ".")
17:     BlogType = BlogTypeArray(BlogTypeArray.Length - 1)
18:     Return BlogType
19: End Function
```

WikiManagement.GetBreadcrumbs Method

This method gets the breadcrumb trail based on the user's navigation selection.

C#

```
public String GetBreadcrumbs();
```

Visual Basic

```
Public Function GetBreadcrumbs() As String
```

Returns

An XHTML unordered list.

Remarks

To generate the breadcrumb trail, this method transforms InformationArchitecture.xml with Breadcrumbs.xslt.

Body Source

```
 1: Public Function GetBreadcrumbs() As String
 2:     Dim XMLUtils As New XMLUtilities
 3:
 4:     'Set XML Args.
 5:     XMLUtils.XMLSourceType = XMLUtilities.XMLSourceTypes.XMLDocument
 6:     XMLUtils.XMLDocument = InformationArchitectureXML
 7:     'Set XSLT Params.
 8:     XMLUtils.XSLTParams.Add("WikiID", Me.wikiID)
 9:     XMLUtils.XSLTParams.Add("Template", Me.Template)
10:     XMLUtils.XSLTParams.Add("TemplateLabel", Me.TemplateLabel)
11:     'Set XSLT Args.
12:     XMLUtils.XSLTSourceType = XMLUtilities.XSLTSourceTypes.File
13:     XMLUtils.XSLTSource = Me.IAPath & "xml/Breadcrumbs.xslt"
14:     'Transform Information Architectre.
15:     GetBreadcrumbs = XMLUtils.TransformXML()
16: End Function
```

WikiManagement.GetInformationArchitecture Method

This method gets the information architecture (navigation/content hierarchy) the current user has permissions to access.

C#

```
public String GetInformationArchitecture();
```

Visual Basic

```
Public Function GetInformationArchitecture() As String
```

Returns

XHTML unordered list representing the Wiki (see page 138) with which the current user has permissions to view/interact.

Remarks

This method crawls the Wiki (see page 138) Management folder tree looking for branches to which the user has permissions to access.

Steps:

1. Represent this structure as XML.

2. Pass this XML to InformationArchitecture.xslt.
3. Return the transformed XML as XHTML unordered list.

Body Source

```
 1: Public Function GetInformationArchitecture() As String
 2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
 3:     Dim InformationArchitectureXHTML As String = ""
 4:
 5:
 6:     'Get Information Architecture.
 7:     If Me.InformationArchitectureXML Is Nothing Then
 8:         InformationArchitectureXML = GetInformationArchitectureXML(False)
 9:     Else
10:         InformationArchitectureXML = Me.InformationArchitectureXML
11:     End If
12:     'Transform Information Architecture XML to XHTML.
13:     If (InformationArchitectureXML IsNot Nothing) Then
14:         InformationArchitectureXHTML = TransformInformationArchitectureXML()
15:     End If
16:     Return InformationArchitectureXHTML
17:
18:
19:
20:
21: End Function
```

WikiManagement.GetInformationArchitectureXML Method

This method generates an XML-based hierarchy of Wiki (see page 138) and its components to which the user has permissions.

C#

```
public System.Xml.XmlDocument GetInformationArchitectureXML(Boolean IncludeMembers);
```

Visual Basic

```
Public Function GetInformationArchitectureXML(ByVal IncludeMembers As Boolean) As
System.Xml.XmlDocument
```

Returns

An XML hierarchy of Wikiand its components.

Remarks

This hierarchy is generated once per-page load.

This hierarchy contains all information necessary to create navigation and gather context for each request.

This accomplishes three main things...

1. All permissions checks are performed once.
2. All folder and content IDs are looked-up once.
3. Context information is made available for each URI.

Body Source

```
 1: Public Function GetInformationArchitectureXML(ByVal IncludeMembers As Boolean) As
System.Xml.XmlDocument
 2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
 3:     Dim WMfolderID As Integer
 4:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 5:     Dim FolderData() As FolderData = Nothing
 6:     Dim DocumentNode, ErrorNode, CurrentNode As System.Xml.XmlNode
 7:     Dim XMLUtils As New XMLUtilities
```

```
 8:     Dim a As New Ektron.Cms.CommonApi
 9:
10:     'Initalize the IA XML document.
11:     DocumentNode = InformationArchitectureXML.CreateElement("ektron")
12:     InformationArchitectureXML.AppendChild(DocumentNode)
13:     CurrentNode = InformationArchitectureXML.DocumentElement
14:
15:     'Get the Wiki management folder's ID.
16:     WMfolderID = Initialize()
17:
18:     'Get all the child folders of the Wiki management folder (these are "Wiki").
19:     Try
20:         FolderData = ContentAPI.GetChildFoldersByFolderId(WMfolderID)
21:     Catch ex As Exception
22:         HttpContext.Current.Response.Redirect("Login.aspx", False)
23:         GetInformationArchitectureXML = Nothing
24:         FolderData = Nothing
25:         Exit Function
26:     End Try
27:
28:     If FolderData IsNot Nothing Then
29:         Try
30:             For Each Folder As FolderData In FolderData
31:                 If ContentAPI.MemberType = 1 Then
32:                     If MembershipUserTraverseOnly(ContentAPI.UserId, Folder.Id) = False
Then
33:                         GetwikiXML(CurrentNode, Folder, InformationArchitectureXML)
34:                     End If
35:                 Else
36:                     GetwikiXML(CurrentNode, Folder, InformationArchitectureXML)
37:                 End If
38:             Next
39:         Catch ex As Exception
40:             ErrorNode = InformationArchitectureXML.CreateElement("li")
41:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, ErrorNode, "status",
"error")
42:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, ErrorNode, "message",
"Could not generate information architecture.")
43:             CurrentNode.AppendChild(ErrorNode)
44:         End Try
45:     End If
46:
47:     If IncludeMembers = True Then
48:         Me.InformationArchitectureXML = InformationArchitectureXML
49:         GetMembersXML()
50:     End If
51:
52:     Return InformationArchitectureXML
53: End Function
```

WikiManagement.GetMembersXML Method

This method inserts member-information to the Wiki (see page 138) nodes in the InformationArchitecture.xml generated by GetInformationArchitectureXML (see page 150)().

C#

```
private System.Xml.XmlDocument GetMembersXML();
```

Visual Basic

```
Private Function GetMembersXML() As System.Xml.XmlDocument
```

Returns

InformationArchitecture.xml including membership users assigned to each Wiki (see page 138).

Remarks

This method inserts a "Members" node to each Wiki (see page 138). Then, loops through the membership group assigned to the Wiki (see page 138) and inserts a "Member" node (with ID, username, firstname, lastname attributes) for each Wiki (see page 138) to the "Members" node.

Body Source

```
 1: Private Function GetMembersXML() As System.Xml.XmlDocument
 2:     Dim InformationArchitectureXML As New System.Xml.XmlDocument
 3:     Dim WikiNodeList As System.Xml.XmlNodeList = Nothing
 4:     Dim Users As New Ektron.Cms.API.User.User
 5:     Dim UserGroupData As New Ektron.Cms.UserGroupData
 6:     Dim Members As New Collection
 7:     Dim MembersNode, MemberNode As System.Xml.XmlNode
 8:     Dim XMLUtils As New XMLUtilities
 9:     Dim MemberData As New UserData
10:     Dim TempID As Integer
11:
12:     InformationArchitectureXML = Me.InformationArchitectureXML
13:     If (InformationArchitectureXML IsNot Nothing) Then
14:         WikiNodeList = InformationArchitectureXML.SelectNodes("//li[@class='Wiki']")
15:     End If
16:     If (WikiNodeList Is Nothing) Then
17:         Return InformationArchitectureXML
18:         Exit Function
19:     End If
20:     For Each Wiki As XmlNode In WikiNodeList
21:         Try
22:             UserGroupData = Users.GetUserGroupByName("wiki." &
Wiki.Attributes.GetNamedItem("label").Value)
23:         Catch ex As Exception
24:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, Wiki, "error",
Wiki.Attributes.GetNamedItem("label").Value)
25:         End Try
26:
27:         If UserGroupData IsNot Nothing Then
28:             'Create a "members" node for this Wiki.
29:             MembersNode = InformationArchitectureXML.CreateElement("li")
30:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MembersNode, "label",
Wiki.Attributes.GetNamedItem("label").Value & " Members")
31:             XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MembersNode, "class",
"members")
32:             Wiki.InsertBefore(MembersNode, Wiki.FirstChild)
33:
34:             'Store the UserID of the actual user in a Temp var.
35:             TempID = Users.RequestInformationRef.CallerId
36:             'Set the UserID internal admin
37:             Users.RequestInformationRef.CallerId =
Ektron.Cms.Common.EkConstants.InternalAdmin
38:             'Get all users that belong to the Wiki's membership group using Internal Admin.
39:             Members = Users.EkUserRef.GetUsersByGroupv2_0(UserGroupData.GroupId, "")
40:
41:             If Members IsNot Nothing Then
42:                 For Each Member As Collection In Members
43:                     MemberData = Users.GetUser(Member.Item("UserID"))
44:                     MemberNode = IADoc.CreateElement("li")
45:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"class", "member")
46:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"id", Member.Item("UserID"))
47:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"username", MemberData.Username)
48:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
"firstname", MemberData.FirstName)
49:                     XMLUtils.ApplyXMLAttribute(InformationArchitectureXML, MemberNode,
```

```
"lastname", MemberData.LastName)
50:                     MembersNode.AppendChild(MemberNode)
51:                 Next
52:             End If
53:
54:             'Reset the UserID to the ID of the actual user.
55:             Users.RequestInformationRef.CallerId = TempID
56:         End If
57:
58:     Next
59:
60:     Return InformationArchitectureXML
61: End Function
```

WikiManagement.GetMetadataDefinitionID Method

This method gets IDs for any of the three metadata definitions for the Wiki (see page 138) Management StarterApp.

C#

```
public int GetMetadataDefinitionID(MetadataDefinitionTypes Type);
```

Visual Basic

```
Public Function GetMetadataDefinitionID(ByVal Type As MetadataDefinitionTypes) As Integer
```

Parameters

Parameters	Description
Type	"Type" enumerated to restrict the lookup to the three required metadata definitions required for this application.

Returns

A single metadata definition ID.

Remarks

This method is intended for use with the Wiki (see page 138) Management StarterApp. This method is not intended to be a general purpose lookup for metadata definition IDs.

Body Source

```
 1: Public Function GetMetadataDefinitionID(ByVal Type As MetadataDefinitionTypes) As Integer
 2:     Dim Metadata As New Metadata
 3:     Dim MetadataCollection() As Ektron.Cms.ContentMetaData
 4:     Dim MetadataType As New Ektron.Cms.ContentMetaData
 5:     Dim SelectedMetadataType As String
 6:
 7:     Select Case Type
 8:         Case MetadataDefinitionTypes.Wiki
 9:             SelectedMetadataType = "wiki.DefaultContent"
10:         Case MetadataDefinitionTypes.CMSBlog
11:             SelectedMetadataType = "wiki.CMSBlog"
12:         Case MetadataDefinitionTypes.MembershipBlog
13:             SelectedMetadataType = "wiki.MembershipBlog"
14:         Case Else
15:             SelectedMetadataType = "None"
16:     End Select
17:
18:     MetadataCollection = Metadata.GetMetaDataTypes("Title")
19:     For Each MetadataType In MetadataCollection
20:         If MetadataType.TypeName = SelectedMetadataType Then
21:             GetMetadataDefinitionID = MetadataType.TypeId
22:             Exit For
23:         Else
24:             GetMetadataDefinitionID = -1
25:         End If
26:     Next
27:
```

```
28:     Return GetMetadataDefinitionID
29: End Function
```

WikiManagement.GetwikiXML Method

This method collects blog component information and inserts this information into InformationArchitecture.xml. This method recursively calls itself to gather all folders/sub-folders of a particular Wiki (see page 138).

C#

```
private GetwikiXML(XmlNode CurrentNode, Ektron.Cms.FolderData FolderData,
System.Xml.XmlDocument IAdoc);
```

Visual Basic

```
Private Sub GetwikiXML(ByRef CurrentNode As XmlNode, ByRef FolderData As
Ektron.Cms.FolderData, ByRef IAdoc As System.Xml.XmlDocument)
```

Parameters

Parameters	Description
CurrentNode	The current folder in the recursive loop.
FolderData	The Ektron FolderData object of the current folder in the recursive loop.
IAdoc	The pointer to the InformationAcrichtecture.xml document being built.

Remarks

This method fills out the details of Wiki (see page 138)-level components. It collects as XML attributes the Wiki (see page 138) or component's...

- **label** - the name of the Wiki (see page 138) or component.
- **id** - the folder ID.
- **class** - enumerated as "Wiki (see page 138)," "Blog," "DiscussionBoard," "DiscussionForum," "Content," "Domain," "Root," or "Community."

*

Body Source

```
 1: Private Sub GetwikiXML(ByRef CurrentNode As XmlNode, ByRef FolderData As
Ektron.Cms.FolderData, ByRef IAdoc As System.Xml.XmlDocument)
 2:     Dim Node, ErrorNode As XmlNode
 3:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 4:     Dim Permissions As New Ektron.Cms.API.Permissions
 5:     Dim XMLUtils As New XMLUtilities
 6:     Dim NodeDepth As Integer
 7:     Dim NodeDepthNavigator As System.Xml.XPath.XPathNavigator
 8:     Dim NodeClass As String = "Undetermined"
 9:     Dim GetBlogType As Boolean = False
10:     Dim BlogType As String
11:
12:     'Create a new li element to hold information regarding the current level in the Info
Architecture.
13:     Node = IAdoc.CreateElement("li")
14:
15:     'Add Title and ID attributes.
16:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "label", FolderData.Name)
17:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "id", FolderData.Id)
18:
19:     'Class the node by virtue of its depth in the hierarchy.
20:     NodeDepthNavigator = CurrentNode.CreateNavigator
21:     NodeDepth = NodeDepthNavigator.Evaluate("count(ancestor-or-self::li)")
22:     Select Case NodeDepth
23:         Case 0
24:             NodeClass = "wiki"
25:
```

```
26:         Case Else
27:             Select Case FolderData.FolderType
28:                 Case 0
29:                     NodeClass = "Content"
30:                 Case 1
31:                     NodeClass = "Blog"
32:                     GetBlogType = True
33:                 Case 2
34:                     NodeClass = "Domain"
35:                 Case 3
36:                     NodeClass = "DiscussionBoard"
37:                 Case 4
38:                     NodeClass = "DiscussionForum"
39:                 Case 5
40:                     NodeClass = "Root"
41:                 Case 6
42:                     Select Case FolderData.Name
43:
44:                         Case "FAQ"
45:                             NodeClass = "CommunityFAQ"
46:                         Case Else
47:                             NodeClass = "Community"
48:                     End Select
49:             End Select
50:     End Select
51:
52:     XMLUtils.ApplyXMLAttribute(IAdoc, Node, "class", NodeClass)
53:
54:     If GetBlogType = True Then
55:         BlogType = GetBlogPermissionsType(FolderData.Id)
56:         XMLUtils.ApplyXMLAttribute(IAdoc, Node, "type", BlogType)
57:     End If
58:
59:     'Append current folder node (with all its attributes, and child node with their
attributes) to its parent node).
60:     CurrentNode.AppendChild(Node)
61:
62:     'Get all child folders that current user has permissions to view.
63:     Dim ChildFolders As New Ektron.Cms.ContentAPI
64:     Dim ChildFolderData() As Ektron.Cms.FolderData = Nothing
65:
66:     Try
67:         ChildFolderData = ChildFolders.GetChildFoldersByFolderId(FolderData.Id)
68:         If ChildFolderData IsNot Nothing Then
69:             For Each ChildFolder As FolderData In ChildFolderData
70:                 If ContentAPI.MemberType = 1 Then
71:                     If MembershipUserTraverseOnly(ContentAPI.UserId, ChildFolder.Id) =
False Then
72:                         GetwikiXML(Node, ChildFolder, IAdoc)
73:                     End If
74:                 Else
75:                     GetwikiXML(Node, ChildFolder, IAdoc)
76:                 End If
77:             Next
78:         End If
79:     Catch ex As Exception
80:         ErrorNode = IAdoc.CreateElement("li")
81:         XMLUtils.ApplyXMLAttribute(IAdoc, ErrorNode, "status", "error")
82:         XMLUtils.ApplyXMLAttribute(IAdoc, ErrorNode, "message", ex.Message)
83:         CurrentNode.AppendChild(ErrorNode)
84:     End Try
85: End Sub
```

WikiManagement.Initialize Method

The Initialize() method is executed upon each page load to ensure StartApp has the components it needs, and returns the "Wiki

(see page 138) Management" folder ID.

C#

```
public int Initialize();
```

Visual Basic

```
Public Function Initialize() As Integer
```

Returns

Wiki (see page 138) Management folder ID.

Remarks

This method...

1. checks to see if the "StarterApps (see page 1)" and "Wiki (see page 138) Management" folders exists, and if not, creates them.
2. checks to see if the "starterapps.WM" CMS (see page 1) user group exists, and if not, creates it.
3. checks to see if the three required metadata values "WM.DefaultContent", "WM.Blog", and exist and if not, creates them.
4. checks to see if the required .aspx templates are registered with CMS400 and if not, registers them.

Body Source

```
Public Function Initialize() As Integer
    Dim FolderData() As FolderData
    Dim StarterAppsFolderID As Integer = -1
    Dim WMFolderID As Integer = -1
    Dim CMSGroupID As Integer = -1
    Dim MemGroupID As Integer = -1

    Dim StarterAppsFolderExists As Boolean = False
    Dim WMFolderExists As Boolean = False
    Dim GroupName As String = ""
    Dim mem_GroupName As String = ""
    Dim WMUser As New StarterApps.Wiki.Users
    Dim Metadata As New Metadata
    Dim MetadataCollection() As Ektron.Cms.ContentMetaData
    Dim MetadataType As New Ektron.Cms.ContentMetaData
    Dim MetadataID As Integer
    Dim UserGroup As New API.User.User
    Dim MembershipBlogMetadataExists, CMSBlogMetadataExists, WikiMetadataExists As Boolean

    'Get all child folders of the CMS Root folder and see if the
    'folder "Starter Apps" has already been created.
    'If so, set StarterAppsFolderExists to TRUE
    FolderData = FolderAPI.GetChildFolders(0, False)
    If Not FolderData Is Nothing Then
        For Each FolderDataItem As FolderData In FolderData
            If FolderDataItem.Name = "Starter Apps" Then
                StarterAppsFolderID = FolderDataItem.Id
                StarterAppsFolderExists = True
                Exit For
            End If
        Next
    End If

    Try
        'If the "Application Accelerators" folder doesn't exist, create it
        If StarterAppsFolderExists = False Then
            StarterAppsFolderID = AddFolder("Starter Apps", 0)
            'Break inheritance and remove inherited permissions
            SetFolderPermissions(StarterAppsFolderID)
            'Remove permissions for all inherited groups
            RemoveInheritedUserGroups(StarterAppsFolderID)
```

```
            'Give everyone traverse-only permission on this folder
            WMUser.AddUserGroupToFolder(StarterAppsFolderID, EVERYONEGROUP, True)

            'Make Folder is marked 'Private'
            SetFolderPrivate(StarterAppsFolderID)
        End If
    Catch ex As Exception
        'clean up
        StarterAppsFolderID = FolderAPI.GetFolderId("Starter Apps", 0)
        If (StarterAppsFolderID <> -1) Then
            FolderAPI.DeleteFolderById(StarterAppsFolderID)
        End If
        If (CMSGroupID <> -1) Then
            UserGroup.DeleteUserGroup(CMSGroupID)
        End If
        Throw ex
        Exit Function
    End Try
    If (StarterAppsFolderID > 0) Then
        'Get all child folders of the "Application Accelerators" folder
        'to see if the "Wiki Management" folder exists.
        FolderData = FolderAPI.GetChildFolders(StarterAppsFolderID, False)
        If Not FolderData Is Nothing Then
            For Each FolderDataItem As FolderData In FolderData
                If FolderDataItem.Name = "Wiki" Then
                    WMFolderID = FolderDataItem.Id
                    WMFolderExists = True
                    Exit For
                End If
            Next
        End If


        'If the WM folder doesn't exist, folder doesn't exist, create it.
        If WMFolderExists = False Then
            WMFolderID = AddFolder("Wiki", StarterAppsFolderID)
            'Break inheritance and remove inherited permissions.
            SetFolderPermissions(WMFolderID)
            'Remove permissions for all inherited groups.
            RemoveInheritedUserGroups(WMFolderID)
            WMUser.AddUserGroupToFolder(WMFolderID, 2, True)
            WMUser.AddUserGroupToFolder(WMFolderID, 888888, True)
            'Make Folder is marked 'Private'.
            SetFolderPrivate(WMFolderID)
        End If

        'See if the metadata mefinition for default content exists.
        'If not, create it.

        MembershipBlogMetadataExists = False
        CMSBlogMetadataExists = False
        WikiMetadataExists = False

        MetadataCollection = Metadata.GetMetaDataTypes("Title")
        For Each MetadataType In MetadataCollection
            Select Case MetadataType.TypeName
                Case "wiki.DefaultContent"
                    MetadataID = MetadataType.TypeId
                    WikiMetadataExists = True
                Case "wiki.MembershipBlog"
                    MetadataID = MetadataType.TypeId
                    MembershipBlogMetadataExists = True
                Case "wiki.CMSBlog"
                    MetadataID = MetadataType.TypeId
                    CMSBlogMetadataExists = True
            End Select
```

```
108:         Next
109:
110:         If WikiMetadataExists = False Then
111:             MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.Wiki)
112:         End If
113:
114:         If MembershipBlogMetadataExists = False Then
115:             MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.MembershipBlog)
116:         End If
117:
118:         If CMSBlogMetadataExists = False Then
119:             MetadataID = AddMetadataDefinition(MetadataDefinitionTypes.CMSBlog)
120:         End If
121:
122:         'Register Page Templates.
123:         RegisterTemplate("WikiHome.aspx")
124:         RegisterTemplate("Wiki.aspx")
125:         RegisterTemplate("Discussion.aspx")
126:         RegisterTemplate("Blog.aspx")
127:         RegisterTemplate("Login.aspx")
128:         RegisterTemplate("Dynamic.aspx")
129:         RegisterTemplate("FAQ.aspx")
130:
131:     End If
132:     Return WMFolderID
133: End Function
```

WikiManagement.MembershipUserTraverseOnly Method

This method checks to see if a Membership user ID has been specifically granted permissions to a folder.

C#

```
private Boolean MembershipUserTraverseOnly(int UserId, int FolderID);
```

Visual Basic

```
Private Function MembershipUserTraverseOnly(ByVal UserId As Integer, ByVal FolderID As
Integer) As Boolean
```

Parameters

Parameters	Description
UserId	The Membership user's ID.
FolderID	The folder ID of the folder we wish to check.

Returns

True - if the user can only Traverse the folder.

False - if the user has been granted permissions to interact (e.g. add, delete, edit...) with the content in the folder.

Remarks

By default, Membership users have Traverse permissions for all folders. This is a hardcoded setting! This method figures out if a Membership user has been specifically granted permission for a folder via direct assignment of the user to the folder, or via a Membership user group that has been assigned to the folder. If the user has permissions beyond Traverse, the return value is set to "False", else it is set to "True."

Body Source

```
 1: Private Function MembershipUserTraverseOnly(ByVal UserId As Integer, ByVal FolderID As
Integer) As Boolean
 2:     Dim UserAPI As New Ektron.Cms.API.User.User
 3:     Dim GroupData() As Ektron.Cms.GroupData
 4:     Dim PermissionsAPI As New Ektron.Cms.API.Permissions
 5:     Dim PermissionsData() As Ektron.Cms.UserPermissionData
 6:
```

```
 7:     'Get all the groups to which the user is assigned.
 8:     GroupData = UserAPI.GetGroupsUserIsIn(UserId, "groupname")
 9:     'Get the permissions data for the folder being examined.
10:     PermissionsData = PermissionsAPI.GetUserPermissions(FolderID, "folder", 0, "")
11:     'Set the default return value - that the Membership user has ONLY traverse permissions.
12:     MembershipUserTraverseOnly = True
13:
14:     For Each PermissionsItem As UserPermissionData In PermissionsData
15:         If PermissionsItem IsNot Nothing Then
16:             'Check to see if the user has been granted access to the folder directly.
17:             If PermissionsItem.UserId = UserId Then
18:                 'In this case, the user has specifically been granted access to the folder.
19:                 'Set return value to false - user has been granted more access
20:                 'than the default traverse-only permission.
21:                 MembershipUserTraverseOnly = False
22:                 Exit For
23:             End If
24:             'Check to see if the user has been granted access to the folder via a
Membership group.
25:             For Each UserGroup As GroupData In GroupData
26:                 If UserGroup.GroupId = PermissionsItem.GroupId Then
27:                     'In this case, the user belongs to a Membership user group that has
28:                     'specifically been granted access to the folder.
29:                     'Set return value to false - user has been granted more access
30:                     'than the default traverse-only permission.
31:                     MembershipUserTraverseOnly = False
32:                     Exit For
33:                 End If
34:             Next
35:         End If
36:         If MembershipUserTraverseOnly = False Then
37:             Exit For
38:         End If
39:     Next
40: End Function
```

WikiManagement.RegisterTemplate Method

This method registers our templates inside CMS400 during Wiki (see page 138) Management StarterApp Initialization.

C#

```
private RegisterTemplate(String TemplateName);
```

Visual Basic

```
Private Sub RegisterTemplate(ByVal TemplateName As String)
```

Parameters

Parameters	Description
TemplateName	The name of the template to resister.

Remarks

Path to the templates is hard coded in this method as "StarterApps (see page 1)/WikiManagement (see page 143)/"

Body Source

```
 1: Private Sub RegisterTemplate(ByVal TemplateName As String)
 2:     Dim TemplateData As New Collection()
 3:     Dim ContentAPI As New Ektron.Cms.ContentAPI
 4:     Dim ExistingTemplates As TemplateData() = ContentAPI.GetAllTemplates("")
 5:     Dim TemplateFileName As String
 6:     Dim TemplateExists As Boolean = False
 7:
 8:     TemplateFileName = "StarterApps/Wiki/" & TemplateName
 9:
10:     'Check to see if template already exists
```

```
    For Each Template As TemplateData In ExistingTemplates
        If Template.FileName = TemplateFileName Then
            TemplateExists = True
            Exit For
        End If
    Next

    'If the template doesn't exist, add it
    If TemplateExists = False Then
        TemplateData.Add(TemplateFileName, "TemplateFileName")
        ContentAPI.EkContentRef.AddTemplatev2_0(TemplateData)
    End If
End Sub
```

WikiManagement.RemoveInheritedUserGroups Method

This method removes any user groups and users assigned to a folder.

C#

```
private RemoveInheritedUserGroups(int FolderID);
```

Visual Basic

```
Private Sub RemoveInheritedUserGroups(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	The ID of the folder from which to remove all user groups.

Remarks

This method is used to "clean up" a folder after breaking content inheritance and before adding new user groups to create a new permissions model.

Notes

This method removes both all users and all user groups.

Body Source

```
Private Sub RemoveInheritedUserGroups(ByVal FolderID As Integer)
    Dim Permissions As New Ektron.Cms.API.Permissions
    Dim PermissionsData() As Ektron.Cms.UserPermissionData

    PermissionsData = Permissions.GetUserPermissions(FolderID, "folder", 0, "")

    If PermissionsData IsNot Nothing Then
        For Each UserGroup As UserPermissionData In PermissionsData
            Select Case UserGroup.GroupId
                Case -1
                    If UserGroup.UserId > 1 Then
                        'This is a user - delete it.
                        Permissions.DeleteItemPermission(UserGroup.UserId, False,
FolderID, EkEnumeration.CMSObjectTypes.Folder)
                    End If
                Case 1
                    'This is the admin group - do nothing.
                Case Else
                    'This is a group - delete it.
                    Permissions.DeleteItemPermission(UserGroup.GroupId, True, FolderID,
EkEnumeration.CMSObjectTypes.Folder)
            End Select
        Next
    End If
End Sub
```

WikiManagement.SetFolderPermissions Method

This method breaks content and metadata inheritance for folders. You may specify which inheritance property you wish to break or specify if you wish to break both.

C#

```
public SetFolderPermissions(int FolderID);
```

Visual Basic

```
Public Sub SetFolderPermissions(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	Folder ID of the folder you wish to break inheritance.

Remarks

Breaking folder inheritance allows us to apply the three required Wiki (see page 138) Management StarterApp metadata types only to the folders that require them. Only Blogs should be assigned the "blogs" metadata types. The "wiki (see page 130)" folder should only be assigned the "default content" metadata type

Body Source

```
1: Public Sub SetFolderPermissions(ByVal FolderID As Integer)
2:     Dim PermissionData As New Ektron.Cms.UserPermissionData
3:     Dim Permissions As New Ektron.Cms.API.Permissions
4:
5:     'Break Folder Inheritance
6:     Permissions.DisableFolderInheritance(FolderID)
7: End Sub
```

WikiManagement.SetFolderPrivate Method

This method sets the folder to private status.

C#

```
public SetFolderPrivate(int FolderID);
```

Visual Basic

```
Public Sub SetFolderPrivate(ByVal FolderID As Integer)
```

Parameters

Parameters	Description
FolderID	Folder ID of the folder to make private.

Remarks

This is a workaround since there is no direct API call to ensure a folder is set to private. To ensure this method executes properly, we must borrow the internal admin's rights by switching from the user's ID who is setting the folder to private to the constant INTERNALADMIN. Then, switch it back before finishing execution. Calling this method without setting the internal admin, even with an authorized CMS (see page 1) user, results in throwing an error. Since this user has already been authenticated to perform this action (creating a Wiki (see page 138) folder), we can use internal admin to avoid this problem.

Body Source

```
1: Public Sub SetFolderPrivate(ByVal FolderID As Integer)
2:     Dim ContentAPI As New Ektron.Cms.ContentAPI
3:     Dim EkContent As Ektron.Cms.Content.EkContent = ContentAPI.EkContentRef
4:     Dim PageData As New Collection
5:     Dim TempID As Integer
6:
7:     'Store the current users's ID in a temprorary integer var.
```

```
 8:     TempID = ContentAPI.RequestInformationRef.CallerId
 9:     'Set the current user's ID to the INTERNALADMIN constant.
10:     ContentAPI.RequestInformationRef.CallerId = Ektron.Cms.Common.EkConstants.InternalAdmin
11:     PageData.Add(FolderID, "ItemID")
12:     PageData.Add("folder", "RequestType")
13:
14:     'EkContent.DisableItemPrivateSettingv2_0(PageData) ' to make public
15:
16:     EkContent.EnableItemPrivateSettingv2_0(PageData) ' to make private
17:
18:     'Set the current user's ID back
19:     ContentAPI.RequestInformationRef.CallerId = TempID
20: End Sub
```

WikiManagement.TransformInformationArchitectureXML Method

This method transforms Information Architecture XML to an XHTML unordered list.

C#

```
public String TransformInformationArchitectureXML();
```

Visual Basic

```
Public Function TransformInformationArchitectureXML() As String
```

Returns

An XHTML unordered list.

Remarks

This method is used to create the Wiki (see page 138) navigation tree that the current user has permissions to traverse.

Body Source

```
 1: Public Function TransformInformationArchitectureXML() As String
 2:     Dim XMLUtils As New XMLUtilities
 3:
 4:     'Set XML Args.
 5:     XMLUtils.XMLSourceType = XMLUtilities.XMLSourceTypes.XMLDocument
 6:     XMLUtils.XMLDocument = InformationArchitectureXML
 7:     'Set XSLT Params.
 8:     XMLUtils.XSLTParams.Add("UserMode", Me.InformationArchitectureUserMode)
 9:     XMLUtils.XSLTParams.Add("WikiID", Me.wikiID)
10:     'Set XSLT Args.
11:     XMLUtils.XSLTSourceType = XMLUtilities.XSLTSourceTypes.File
12:     XMLUtils.XSLTSource = Me.IAPath & "xml/InformationArchitecture.xslt"
13:     'Transform Information Architectre.
14:     TransformInformationArchitectureXML = XMLUtils.TransformXML()
15: End Function
```

WikiManagement Properties

The properties of the WikiManagement class are listed here.

Public Properties

	Name	Description
	IAPath (see page 163)	The IAPath Property indicates the location of InformationArchitecture.xslt.
	InformationArchitectureUserMode (see page 163)	This Property is used to determine "action" rights for users and is consumed by InformationArchitecture.xslt when called by the "TransformInformationArchitetureXML()." Default value is "MembershipUser."
	InformationArchitectureXML (see page 163)	The InformationArchitectureXML property is used to hold the the information architecture prior to being transformed with XSLT.
	Template (see page 164)	The Template property is used in several XSLT transformations to identify the aspx template associated with a link.
	TemplateLabel (see page 164)	The TemplateLabel property indicates the text to display for an aspx template associated with a particular link. This property is consumed by several XSLT transformations.

	wikiID (see page 164)	The WikiID property is the FolderID of the selected Wiki (see page 138) and is consumed by several methods in the WikiManagement (see page 143) namespace.

Legend

	Property

WikiManagement.IAPath Property

The IAPath Property indicates the location of InformationArchitecture.xslt.

C#

```
public String IAPath;
```

Visual Basic

```
Public Property IAPath() As String
```

Returns

Relative location path - e.g. "../"

Description

String

Remarks

The Information Architecture XSLT is called from several physical locations; site root, and from AJAX components under "/actions." This property allows a method calling this XSLT to identify its relative location to InformationArchitecture.xslt.

WikiManagement.InformationArchitectureUserMode Property

This Property is used to determine "action" rights for users and is consumed by InformationArchitecture.xslt when called by the "TransformInformationArchitetureXML()." Default value is "MembershipUser."

C#

```
public InformationArchitectureUserModes InformationArchitectureUserMode;
```

Visual Basic

```
Public Property InformationArchitectureUserMode() As InformationArchitectureUserModes
```

WikiManagement.InformationArchitectureXML Property

The InformationArchitectureXML property is used to hold the the information architecture prior to being transformed with XSLT.

C#

```
public System.Xml.XmlDocument InformationArchitectureXML;
```

Visual Basic

```
Public Property InformationArchitectureXML() As System.Xml.XmlDocument
```

Returns

The Information Architecture XmlDocument.

Description

XmlDocument

Remarks

IA XML is transformned in several places - to generate the Wiki (see page 138) list, to get the breadcrumbs, to fill out links in the pWiki navigation widget.

WikiManagement.Template Property

The Template property is used in several XSLT transformations to identify the aspx template associated with a link.

C#

```
public String Template;
```

Visual Basic

```
Public Property Template() As String
```

Returns

The ASPX template associated with a link.

Description

String

Remarks

This is the template target of a link. See property TemplateLabel (see page 164)().

WikiManagement.TemplateLabel Property

The TemplateLabel property indicates the text to display for an aspx template associated with a particular link. This property is consumed by several XSLT transformations.

C#

```
public String TemplateLabel;
```

Visual Basic

```
Public Property TemplateLabel() As String
```

Returns

The display string for a link template.

Description

String

Remarks

This label is what shows up to the user.

WikiManagement.wikiID Property

The WikiID property is the FolderID of the selected Wiki (see page 138) and is consumed by several methods in the WikiManagement (see page 143) namespace.

C#

```
public int wikiID;
```

Visual Basic

```
Public Property wikiID() As Integer
```

Returns

FolderID of the selected Wiki (see page 138).

Remarks

This is used all over the place.

XMLUtilities Class

Contains methods and properties for handling the XML used within the Wiki (see page 138) Starter Application.

Class Hierarchy

C#

```
public class XMLUtilities;
```

Visual Basic

```
Public Class XMLUtilities
```

File

Wiki.vb

XMLUtilities Enumerations

The enumerations of the XMLUtilities class are listed here.

Enumerations

	Name	Description
	XMLSourceTypes (see page 165)	Sets the input type of the XML to be transformed.
	XSLTSourceTypes (see page 165)	Sets the input type of the XSLT to be used as the transforming document.

Legend

	Enumeration

Ektron.Cms.StarterApps.Wiki.XMLUtilities.XMLSourceTypes Enumeration

Sets the input type of the XML to be transformed.

C#

```
public enum XMLSourceTypes {
}
```

Visual Basic

```
Public Enum XMLSourceTypes
End Enum
```

File

Wiki.vb

Remarks

If the type is set to XMLDocument (see page 170), you must set the XMLDocument (see page 170) property to the XMLDocument (see page 170) you wish to transform. Otherwise, set the the XMLSource (see page 170) property to the string location of the XMLDocument (see page 170) you wish to transform.

Ektron.Cms.StarterApps.Wiki.XMLUtilities.XSLTSourceTypes Enumeration

Sets the input type of the XSLT to be used as the transforming document.

C#

```
public enum XSLTSourceTypes {
```

```
}
```

Visual Basic

```
Public Enum XSLTSourceTypes
End Enum
```

File

Wiki.vb

Remarks

You must also set the the XSLTSource (see page 171) property to the string location of the XSLTDocument you wish to use as the transformation document.

XMLUtilities Methods

The methods of the XMLUtilities class are listed here.

Public Methods

	Name	Description
	ApplyXMLAttribute (see page 166)	This method applies XML attributes to an XML Node.
	PrettyPrintXML (see page 167)	Returns the contents of XMLDocument (see page 170) nicely indented.
	TransformXML (see page 167)	This method is a general purpose XML transformation method. This method allows arguments to be passed to the XSLT transformation.

Legend

	Method

XMLUtilities.ApplyXMLAttribute Method

This method applies XML attributes to an XML Node.

C#

```
public ApplyXMLAttribute(XmlDocument IADoc, XmlNode Node, String AttributeName, String
AttributeValue);
```

Visual Basic

```
Public Sub ApplyXMLAttribute(ByRef IADoc As XmlDocument, ByRef Node As XmlNode, ByVal
AttributeName As String, ByVal AttributeValue As String)
```

Parameters

Parameters	Description
IADoc	The XML document being parsed.
Node	The node in which to add the attribute.
AttributeName	The attribute's name.
AttributeValue	The attribute's value.

Remarks

This is a general purpose method meant to simplify adding attributes to nodes. It's meant to help keep code tidy.

Body Source

```
1: Public Sub ApplyXMLAttribute(ByRef IADoc As XmlDocument, ByRef Node As XmlNode, ByVal
AttributeName As String, ByVal AttributeValue As String)
2:     Dim Attribute As XmlAttribute
3:     Attribute = IADoc.CreateAttribute(AttributeName)
4:     Attribute.InnerText = AttributeValue
5:     Node.Attributes.Append(Attribute)
6: End Sub
```

XMLUtilities.PrettyPrintXML Method

Returns the contents of XMLDocument (see page 170) nicely indented.

C#

```
public String PrettyPrintXML(System.Xml.XmlDocument XMLDocument);
```

Visual Basic

```
Public Function PrettyPrintXML(ByRef XMLDocument As System.Xml.XmlDocument) As String
```

Parameters

Parameters	Description
XMLDocument	The XMLdocument to format.

Returns

Formatted string.

Remarks

Used primarily to make long XML/XHTML readable.

Body Source

```
 1: Public Function PrettyPrintXML(ByRef XMLDocument As System.Xml.XmlDocument) As String
 2:     'Create stream in which to load the formatted XML document.
 3:     Dim OutputStream As System.IO.Stream = New System.IO.MemoryStream
 4:
 5:     'Create XMLTextWriter with formatting specifics, and direct output into OutputStream.
 6:     Dim MyXmlTextWriter As New System.Xml.XmlTextWriter(OutputStream, Nothing)
 7:     MyXmlTextWriter.Formatting = Formatting.Indented
 8:     MyXmlTextWriter.Indentation = 5
 9:     MyXmlTextWriter.IndentChar = " "
10:
11:     'Load the contents of XMLDocument argument into XMLTextWriter.
12:     XMLDocument.Save(MyXmlTextWriter)
13:
14:     'Load Formatted XML Stream into StreamReader.
15:     Dim XMLStreamReader As New System.IO.StreamReader(OutputStream)
16:     OutputStream.Flush()
17:     OutputStream.Position = 0
18:
19:     'Return Indented XML String.
20:     PrettyPrintXML = XMLStreamReader.ReadToEnd()
21:     XMLStreamReader.Close()
22: End Function
```

XMLUtilities.TransformXML Method

This method is a general purpose XML transformation method. This method allows arguments to be passed to the XSLT transformation.

C#

```
public String TransformXML();
```

Visual Basic

```
Public Function TransformXML() As String
```

Returns

XML transformed by an XSLT stylesheet.

Remarks

This method takes XML source from

1. A string of xml text.
2. A URI (http://something).
3. A file on the file system.
4. An object of type system.xml.xmldocument.

To take advantage of the first three XML source type options, you must:

a. Set the XMLSourceType (see page 171) property to XMLString, URI, or File.

b. Set the XMLSource (see page 170) property to the string containing the corresponding value.

To take advantage of the fourth XML source type, you must:

a. Set the XMLSourceType (see page 171) property to XMLDocument (see page 170).

b. Set the XMLDocument (see page 170) property to the XML document you wish to transform.

This method takes XSLT source from

1. A string of xml text.
2. A URI (http://something).
3. A file on the file system.

To set the XSLT source type options, you must:

a. Set the XSLTSourceType (see page 172) property to XMLString, URI, or File.

b. Set the XMLSource (see page 170) property to the string containing the corresponding value.

This method takes an arbitrary number of paramaters to pass into the XSLT transformation. In order to pass params into the XSLT transform, you must:

a. Populate a NameValueCollection (param name, param, value).

b. Set the XSLTParams (see page 171) property to your populated NameValueCollection.

Body Source

```
 1: Public Function TransformXML() As String
 2:     Dim myXML As New System.Xml.XmlDocument
 3:     Select Case Me.XMLSourceTypeValue
 4:         Case XMLSourceTypes.File
 5:             Try
 6:                 myXML.Load(System.Web.HttpContext.Current.Server.MapPath(XMLSourceValue))
 7:             Catch ex As Exception
 8:                 Return "Cannot Load XML File: " & Me.XMLSource
 9:             End Try
10:         Case XMLSourceTypes.URI
11:             Try
12:                 myXML.Load(XMLSourceValue)
13:             Catch ex As Exception
14:                 Return "Cannot Load XML File: " & Me.XMLSource
15:             End Try
16:         Case XMLSourceTypes.XMLString
17:             Try
18:                 myXML.LoadXml(XMLSourceValue)
19:             Catch ex As Exception
20:                 Return "Cannot Load XML String: " & ex.Message
21:             End Try
```

```
22:         Case XMLSourceTypes.XMLDocument
23:             Try
24:                 myXML = Me.XMLDocument
25:             Catch ex As Exception
26:                 Return "Cannot Load XML Document: " & ex.Message
27:             End Try
28:         Case Else
29:             Try
30:                 Throw New Exception("XML Type Property (File, URI, XMLString, XMLDocument)
Not Set")
31:             Catch ex As Exception
32:                 Return "Cannot Load XML: " & ex.Message
33:             End Try
34:     End Select
35:
36:     Dim myXSLT As New System.Xml.Xsl.XslCompiledTransform
37:     Select Case Me.XSLTSourceTypeValue
38:         Case XSLTSourceTypes.File
39:             Try
40:                 myXSLT.Load(System.Web.HttpContext.Current.Server.MapPath(XSLTSourceValue))
41:             Catch ex As Exception
42:                 Return "Cannot Load XSLT File: " & Me.XSLTSource
43:             End Try
44:         Case XSLTSourceTypes.URI
45:             Try
46:                 myXSLT.Load(XSLTSourceValue)
47:             Catch ex As Exception
48:                 Return "Cannot Load XSLT File: " & Me.XSLTSource
49:             End Try
50:         Case XSLTSourceTypes.XSLTString
51:             Try
52:                 Dim myXSLTString As New XmlDocument()
53:                 myXSLTString.LoadXml(XSLTSourceValue)
54:                 myXSLT.Load(myXSLTString)
55:             Catch ex As Exception
56:                 Return "Cannot Load XSLT String: " & ex.Message
57:             End Try
58:         Case Else
59:             Try
60:                 Throw New Exception("XSLT Type Property (File, URI, XSLTString) Not Set")
61:             Catch ex As Exception
62:                 Return "Cannot Load XSLT: " & ex.Message
63:             End Try
64:     End Select
65:
66:     Dim myOutputStream As New System.IO.MemoryStream()
67:     Dim myXSLTArgs As New System.Xml.Xsl.XsltArgumentList
68:
69:     If Not myXSLTParams Is Nothing Then
70:         Dim i As Integer
71:         For i = 0 To Me.XSLTParams.Count - 1
72:             myXSLTArgs.AddParam(Me.XSLTParams.GetKey(i), "", Me.XSLTParams.GetValues(i)(0))
73:         Next
74:         Try
75:             myXSLT.Transform(myXML, myXSLTArgs, myOutputStream)
76:         Catch ex As Exception
77:             Return ex.Message
78:         End Try
79:     Else
80:         Try
81:             myXSLT.Transform(myXML, Nothing, myOutputStream)
82:         Catch ex As Exception
83:             Return ex.Message
84:         End Try
85:     End If
86:
```

```
87:     myOutputStream.Flush()
88:     myOutputStream.Position = 0
89:
90:     Dim myXMLStreamReader As New System.IO.StreamReader(myOutputStream)
91:     Dim myTransformedXML As String
92:
93:     myTransformedXML = myXMLStreamReader.ReadToEnd()
94:     myXMLStreamReader.Close()
95:
96:     Return myTransformedXML
97:
98: End Function
```

XMLUtilities Properties

The properties of the XMLUtilities class are listed here.

Public Properties

	Name	Description
	XMLDocument (see page 170)	Holds the XMLDocument to be transformed.
	XMLSource (see page 170)	If XMLSourceType (see page 171) is NOT set to XMLDocument (see page 170), this property holds the location of the XMLDocument (see page 170) to be transformed.
	XMLSourceType (see page 171)	Holds the type of input XML document.
	XSLTParams (see page 171)	Specifies any parameters to be passed into the XSLT transformation.
	XSLTSource (see page 171)	Specifies where the XSLT document is located.
	XSLTSourceType (see page 172)	Sets the type of document to be used as transformation source - specifies where this document lives.

Legend

	Property

XMLUtilities.XMLDocument Property

Holds the XMLDocument to be transformed.

C#

```
public XmlDocument XMLDocument;
```

Visual Basic

```
Public Property XMLDocument() As XmlDocument
```

Returns

XmlDocument

Remarks

Only needs to be populated if the XMLSourceType (see page 171) is set to XmlDocument.

XMLUtilities.XMLSource Property

If XMLSourceType (see page 171) is NOT set to XMLDocument (see page 170), this property holds the location of the XMLDocument (see page 170) to be transformed.

C#

```
public String XMLSource;
```

Visual Basic

```
Public Property XMLSource() As String
```

Returns

The location of the XML Document to be transformed.

Remarks

Only needs to be populated if the XMLSourceType (see page 171) is not set to XMLDocument (see page 170).

XMLUtilities.XMLSourceType Property

Holds the type of input XML document.

C#

```
public XMLSourceTypes XMLSourceType;
```

Visual Basic

```
Public Property XMLSourceType() As XMLSourceTypes
```

Returns

XML Document source type - XMLString, URI, File, or XMLDocument (see page 170).

Remarks

If the type is set to XMLDocument (see page 170), you must set the XMLDocument (see page 170) property to the XMLDocument (see page 170) you wish to transform. Otherwise, set the the XMLSource (see page 170) property to the string location of the XMLDocument (see page 170) you wish to transform.

XMLUtilities.XSLTParams Property

Specifies any parameters to be passed into the XSLT transformation.

C#

```
public NameValueCollection XSLTParams;
```

Visual Basic

```
Public Property XSLTParams() As NameValueCollection
```

Returns

List of parameters to be passed into the transform

Remarks

Optional - can hold any number of parameters. Specified as "param name", "value." You must define these parameters in your XSLT file to use them.

XMLUtilities.XSLTSource Property

Specifies where the XSLT document is located.

C#

```
public String XSLTSource;
```

Visual Basic

```
Public Property XSLTSource() As String
```

Returns

Location of the XSLT document.

XMLUtilities.XSLTSourceType Property

Sets the type of document to be used as transformation source - specifies where this document lives.

C#

```
public XSLTSourceTypes XSLTSourceType;
```

Visual Basic

```
Public Property XSLTSourceType() As XSLTSourceTypes
```

Returns

String

Remarks

Specifies how to load the XSLT document to be used as transformation source.

Index

C

E

W

X