PRIMAVERA WebCentral SDK Manual

Version 1.0

October 2010

Index

Introduction

This documentation is part of the PRIMAVERA WebCentral Software Development Kit (SDK), which allows developers to build and implement customized components which integrate seamlessly to the PRIMAVERA WebCentral Platform.

The PRIMAVERA WebCentral SDK was in particular targeted for the PRIMAVERA partner community, to enable and speed up product extensibility and customization for their end-users.

The PRIMAVERA WebCentral SDK consists of various utilities which help developers to build state-of-the-art components for the PRIMAVERA WebCentral Platform and publish them to the production environment.

The PRIMAVERA WebCentral SDK can be also a complement of Application Builder solution.

In Application Builder it is possible to build an entity model. After publishing this model, code is auto-generated and made available, in the PRIMAVERA WebCentral structure. However, with this tool it is not possible to customize forms at a layout level and make business rules.

In this document you will find valuable background information, a step-by-step example of how to build a customized component for the PRIMAVERA WebCentral Platform and a reference of the most common *gotchas*. It will provide you a guide and reference whenever you plan to develop customized components to extent the out-of-the-box features of PRIMAVERA WebCentral.

PKB

How to develop a Connector? (Application Builder)
How to develop a Connector? (Workflow Designer)

Overview

Chapter 1 – Gives a general, technical overview about the product itself. Here you will find information about the product structure, installation, operation and the SDK utilities.

Chapter 2 – Step-by-Step example on how to build a module for PRIMAVERA WebCentral.

Chapter 3 – Discusses the integration of the PRIMAVERA ERP.

Chapter 4 – Frequently Asked Questions.

Chapter 5 – Source Code used to build the step-by-step example.

PRIMAVERA WebCentral

In a time of digitalization and optimization of business processes, the use of internet and intranet is getting more and more important. Organizations are searching solutions to easily share information between various communities, such as employees, partners and clients, which for themselves reside in very different locations. Information need to be available 24 hours and 7 days a week at any possible locality.

The PRIMAVERA WebCentral provide a solution for this problem. It implements a community based security model which increases productivity and prevents from a disorganized way of searching information. It provides components that promote employees, partners and clients to quickly solve their daily tasks and problems, such as marking employee holidays, enter organizational requests, dispense records, ordering products, etc, with direct connection to the PRIMAVERA ERP solution. It provides customizable, multi-level workflows for approvation as well as support for any language that you might want to target your portal to.

PRIMAVERA WebCentral is a platform for **content management**, which facilitates the creation of various portals, each orientated to their specific audience, sharing information targeted to any of these audiences but always having an easy way to administrate this information, like events, messages, press-releases, etc.

PRIMAVERA WebCentral is a platform for **collaboration**. It includes features to promote automatization, business workflow and approvation based on a web interface.

PRIMAVERA WebCentral is a platform for **integration** of any kind of content, including components that were developed by third party companies to extent existing components and help managing business processes using the internet or intranet as communication channel to reach employees, clients or partners.

Contents

We understand the content as **output** of a component's execution. The output allows the user to browse through existing content (for example **visualization** of messages), or manage new or already existing content (content **administration**: for example inserting a new message to be shown). As for PRIMAVERA, there is no better definition for content as **everything is content**, since it is managed by components that are compatible with the PRIMAVERA WebCentral Platform.

Components

Understand **Components** as small applications that generate new -and visualize existing contents. Imagination is the only limitation for developing new components. Imagine a component that obtains the current weather report for a selected city, using a Web service and displays this information in the client's browser, or a component that allows to list accounting

details for a customer we selected but also components to introduce, publish and visualize press communications referring to your company.

Components can be designed and developed by any person that has knowledge of how to develop web applications using the Microsoft .NET Framework. This is by far the only and most important requirement needed for developing components for the PRIMAVERA WebCentral Platform.

Modules

Modules that are developed for the PRIMAVERA WebCentral Platform expose various components. Most probably, in the beginning, you'll have difficulties distinguish between **Modules** and **Components**, but soon you'll understand that a module is one Visual Studio Solution, which exposes various components for the PRIMAVERA WebCentral Platform.

Take, for example, the base solution of PRIMAVERA WebCentral and you'll see the following modules, each one exposing categorized components:

Administration	Components to administrate PRIMAVERA WebCentral (security, categories, approvals, etc.)
Productivity	Components that will help you to quickly design your portals pages, such as multimedia components, HTML components, downloads, events, forms, polls, etc.
Human Resources	Components that can be used to connect PRIMAVERA WebCentral to the PRIMAVERA ERP and implement an employee "self-service", for example to mark holidays, print salary receipts, government tax declaration, etc.
Account Receivables	Components that can be used to connect PRIMAVERA WebCentral to the PRIMAVERA ERP and show account receivable information for a client.
Sales	Components, connected to the PRIMAVERA ERP that show sells information as well as the B2B shopping-cart component.

A Technical Overview

The installation directory

The PRIMAVERA WebCentral Platform is normally installed and configured in one of the default Internet Information Services (IIS) directories under the **c:\inetpub\wwwroot\WebCentral** folder. You can find the following type of files in the installation directory:

\Bin	Contains all assemblies that are necessary to run the web based user interface for the PRIMAVERA WebCentral Platform.
\Modules	Contains all registered modules for the PRIMAVERA WebCentral Platform. There are several subfolders, each subfolder contains all necessary files for the module (dll, html, aspx, ascx, etc)
\UserFiles	All files that are uploaded by users end up here. There is a general division of images and documents – each file type has its own folder.

\SystemFiles	As the name already says, this folder contains system files that are not directly used by the PRIMAVERA WebCentral Platform, such as commonly used images, java script files and email notification templates.
\WebTemplates	Templates that define the visual appearance of the PRIMAVERA WebCentral. Here you will find various predefined templates which consist of custom style sheets (CSS) and images. You also can create new templates to customize the portals appearance, based on one of these templates.
\Tools	Various helpful tools such as the instance configuration tool, site administrator and module registrar.

Files like custom style sheets and HTML, ASPX and ASCX files can freely be modified to change the websites behaviour or appearance, but please consider backing them up as they will be overwritten with any future installation or update.

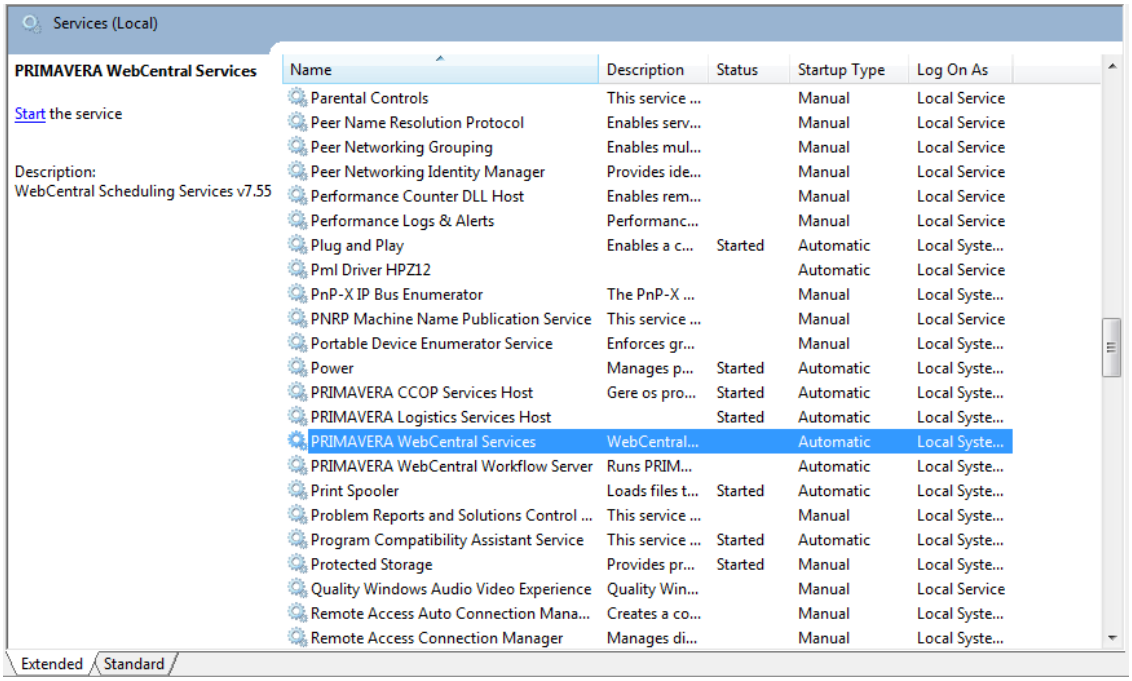
The Database

All portal information, such as web pages, components, security roles, approvals, contents are stored in a SQL Server database. This is the reason why you should create a database management plan to regularly backup, reindex and defragment this database. An optimized database often can speed up the rendering of a webpage by 100% and you'll always have the benefit of a backed up database. Consult the SQL Server Books Online on how to create a Database Management Plan for your database.

The name of the database used by PRIMAVERA WebCentral is **ePrimavera** by default. If you are running more than one PRIMAVERA WebCentral instance, the non-default instance database will be named **ePrimavera_xpto** where **xpto** stands for the name of the instance.

The PRIMAVERA WebCentral Services

Time consuming tasks, like for example broadcasting notifications, are implemented by the PRIMAVERA WebCentral Services. You can control the PRIMAVERA WebCentral Services by opening the system's Management Console (**Control Panel** | **Administrive Tools** | **Services** | **PRIMAVERA WebCentral Services)**.



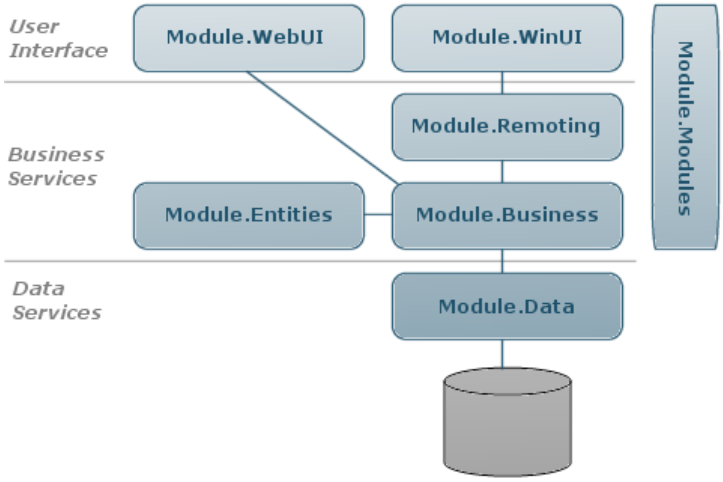
You should stop and disable this service on your development machine whenever not needed, as it uses resources that could block some files from being deployed.

A Developer's View

The PRIMAVERA WebCentral modules are essential parts for developing components for the PRIMAVERA WebCentral Platform. One module exposes the following functionalities for the PRIMAVERA WebCentral general architecture:

- Components that implement certain functionality and can be placed and rendered by one or many portals pages.
- A module can implement functionality which can be inherited and used by other existing modules or components.
- A module or component can use all or parts of the functionality offered by the PRIMAVERA WebCentral Platform (such as security, mappings and approvals).
- Modules group and categorize components that are inter-connected with them selves in a way that a user can find them easily.

A single module, which exposes one or more components for the PRIMAVERA WebCentral, consists of one single Visual Studio 2008 solution having 8 or 9 Visual Studio 2008 projects, which are organized the following way:



WebUI	Web-based user interfaces. These User Controls are usually used to render contents for the client interface (HTML)
WinUI	Windows Forms based user-interfaces. These dialogs are usually used for content or component administration purposes.
IRemoting	Remoting Interfaces (see also Remoting)
Remoting	Remoting Layer. Windows Forms based user-interfaces use remoting to communicate between Client and Server. These classes *implement* the contract defined by the remoting interfaces. They generally to map the implemented methods to the Business Layer.
Entities	Business Entity Layer. This project should contain all entity classes used throughout the module. Entity classes should only expose properties – no business logic (public methods) should be implemented here.
Business	Business Layer. All business rules should be implemented here.
Data	Data Layer. This layer implements the data access to the specified database management system. It is best practise to locate all methods that perform direct data access in this Visual Studio project.
Modules	PRIMAVERA WebCentral Contract project. This project contains classes that interact with the PRIMAVERA WebCentral and exposes information about your module and your module's components, approvals, categories, etc, to the PRIMAVERA WebCentral Platform.
Workflow	This project exposes the events and activities (Create, Remove, Update and Delete) associated to the Business Entity Layer. This connector can be used by Workflow Designer solution to create processes.

User Interface Layer

The user interface layer consists of two .NET assemblies targeting the web and the windows forms user interface (WebUI and WinUI). Whenever possible you should consider using HTML user interfaces to visualize and manage components, as they are less error-prone due configuration problems, easier to develop and target all operating system platforms. However, you might need to implement Window Forms user interfaces to configure the components appearance in the Site Adminstration utility. For example, by the time you place a component onto a portal's page, optionally you can ask the user to configure its appearance – this configuration needs to be done as Windows Forms Dialog.

The WebUI project implements all components that are shown in the client's browser, or let's say HTML. These can be any kind of component, like *managing* the content or *visualizing* the content. Web-based component's are usually implemented as plain ASP.NET user control (ascx), but inherit specific methods from the PRIMAVERA WebCentral Platform base class **`WebComponentBase`**.

Remoting Layer

The remoting layer, which is implemented by the assemblies **IRemoting** and **Remoting**, is necessary to achieve communications between client and server whenever working with Windows Forms-based user interfaces. In this case, the user interface is hosted in the browser (like an ActiveX control) on the client machine, whereas the engine (business layer) is running on the web server. Remoting is used to *tunnel* all communications between client user interface and servers business layer.

The IRemoting assembly defines the contract between the code that runs on the client and the code that runs on the web server. The actual implementation of this contract is only available on the web server. The following segment shows a typical example of how to code a components interface:

```
Public Interface ICompFlashPlayers
    Function Exists(ByVal RemoteContext As RemoteEngineContext, _
        ByVal ID As System.Guid) As System.Boolean

    Function Edit(ByVal RemoteContext As RemoteEngineContext, _
        ByVal ID As System.Guid) As Entities.CompFlashPlayer

    Function Clone(ByVal RemoteContext As RemoteEngineContext, _
        ByVal ID As System.Guid) As System.Guid

    Sub Update(ByVal RemoteContext As RemoteEngineContext, _
        ByRef CompFlashPlayer As Entities.CompFlashPlayer)

    Sub Remove(ByVal RemoteContext As RemoteEngineContext, _
        ByVal ID As System.Guid)
End Interface
```

The **Remoting** assembly **implements** the contract **IRemoting** and will be run on the web server and generally maps to methods implemented by the **Business** project of the module. A typical implementation of a remoting component is shown by the following code segment:

```
    Public Class CompFlashPlayers
        Inherits Engine.Remoting.RemoteServiceBase
        Implements ICompFlashPlayers

        Public Function Clone(ByVal RemoteContext As
Engine.IRemoting.RemoteEngineContext, _
            ByVal ID As System.Guid) As System.Guid
Implements IRemoting.ICompFlashPlayers.Clone
            Dim BSO As New Business.CompFlashPlayers
            Return BSO.Clone(RemoteContext.EngineContext, ID)
        End Function

        Public Function Edit(ByVal RemoteContext As
Engine.IRemoting.RemoteEngineContext, _
            ByVal ID As System.Guid) As Entities.CompFlashPlayer
Implements IRemoting.ICompFlashPlayers.Edit
            Dim BSO As New Business.CompFlashPlayers
            Return BSO.Edit(RemoteContext.EngineContext, ID)
        End Function

        Public Function Exists(ByVal RemoteContext As
Engine.IRemoting.RemoteEngineContext, _
            ByVal ID As System.Guid) As Boolean
Implements IRemoting.ICompFlashPlayers.Exists
            Dim BSO As New Business.CompFlashPlayers
            Return BSO.Exists(RemoteContext.EngineContext, ID)
        End Function

        Public Sub Remove(ByVal RemoteContext As Engine.IRemoting.RemoteEngineContext,
_
            ByVal ID As System.Guid) Implements IRemoting.ICompFlashPlayers.Remove
            Dim BSO As New Business.CompFlashPlayers
            BSO.Remove(RemoteContext.EngineContext, ID)
        End Sub

        Public Sub Update(ByVal RemoteContext As Engine.IRemoting.RemoteEngineContext,
_
            ByRef CompFlashPlayer As Entities.CompFlashPlayer)
Implements IRemoting.ICompFlashPlayers.Update
            Dim BSO As New Business.CompFlashPlayers
            BSO.Update(RemoteContext.EngineContext, CompFlashPlayer)
        End Sub
    End Class
```

Business Entities, Business Services and Data Services

The Business Entities, Business Services and Data Services assemblies implement the 3-tier layers that generally are used in all PRIMAVERA Applications.

PRIMAVERA WebCentral Modules Contract Layer

The Modules assembly contains classes that identify the module as developed for the PRIMAVERA WebCentral Platform and exports contractual information about the module as well as all implemented components.

The following information will be implemented in the modules assembly:

- Information about the module (name, description, author, version)
- All components implemented by this module as well as their attributes
- Category types used for the implemented components (optional)
- Properties for the approval rules (optional)
- Mapping information to integrate with the PRIMAVERA ERP (optional)

Some parts of these definitions are required and must follow the specified rules to successfully integrate the module to the PRIMAVERA WebCentral Platform (for example, information about the module) – other parts are optional and only need to be considered if necessary.

PRIMAVERA WebCentral SDK

The PRIMAVERA WebCentral SDK's goal is to facilitate the conception, development and maintenance of customized modules for the PRIMAVERA WebCentral. This product was conceived especially, to enable partners to customize and extend the WebCentral base (or the model developed in Application Builder) solution and find an approach to all the specific requirements and needs for the final client. You should consider PRIMAVERA WebCentral as an open platform to integrate new modules in a simple and efficient way, and the PRIMAVERA WebCentral SDK is the *key* to that. The WebCentral SDK consists of:

- A detailed documentation which outlines the concepts of the PRIMAVERA WebCentral and a step-by-step introduction on how to build customize modules.
- A set of templates for explaining how to use the most important functionality exposed by the PRIMAVERA WebCentral (for example: security, approvals and categories).
- A set of tools that are used by PRIMAVERA to develop and deploy customized solutions that integrates with PRIMAVERA WebCentral.

Requirements

Before starting designing and developing components for the PRIMAVERA WebCentral you should be aware of the minimum requirements to be met:

Hardware Requirements

- Intel or compatible Pentium 4 (1600 MHz or above)
- 1024 x 768 screen resolution
- 2048 memory of RAM
- 1GB space on the hard disk

Software Requirements

- Microsoft Windows XP, Microsoft Windows 2003 Server or newer
- Microsoft Internet Information Services 6 or newer
- Microsoft Visual Studio 2008 and NET Framework 3.5 SP1

- Microsoft SQL Server 2000 or newer

Please note that the Internet Information Services (IIS) should be installed before installing the Microsoft .NET Framework. Also note that if using IIS 7 or newer you **must** select the following options from the IIS install components:

Other Resources

Other additional important resources on how to develop Microsoft ASP.NET solutions are:

Microsoft ASP.NET	http://www.asp.net/
Microsoft SQL Server	http://www.microsoft.com/sql/
Microsoft SQL Server Developer Center	http://msdn.microsoft.com/sqlserver/
Microsoft Visual Basic Developer Center	http://msdn.microsoft.com/vbasic/
Microsoft Developer Network	http://msdn.microsoft.com/

PRIMAVERA WebCentral SDK

The PRIMAVERA WebCentral SDK is installed automatically with PRIMAVERA WebCentral setup. The installation process copy files to the following folder:

\SDK	Assemblies for the WebCentral Platform

PKB

Installation Manual - WebCentral

Step-by-Step

This chapter will explain – step by step – how to create a module for the PRIMAVERA WebCentral Platform. The module we will create during this chapter contains various components and should serve you as introduction on how to create your own module for your own specific needs.

Step 1 – Creating a new module

Step 1.1 – Creating the base solution

By default, PRIMAVERA WebCentral have available one project, "Primavera.BusinessModel", that we can deploy under Application Builder. But we can create more projects and also use the Application Builder.

Like referred earlier in this manual, a base solution for a module consist of 8 projects (can go up to 10 if Workflow and ERM connectors are to be included).

PKB

Application Builder

Step 1.2 – Signing the Assemblies

All assemblies for the PRIMAVERA WebCentral Platform must be correctly signed using the strong name (**sn.exe**) utility from the Visual Studio 2008 Command Prompt. By inserting the attributes **AssemblyKeyFile**, **AssemblyKeyName** e **AssemblyDelaySign** to the file **AssemblyInfo.vb** of each solution's project, the PRIMAVERA WebCentral SDK Addin already prepared all generated assemblies to be signed by adding the attributes:

```
#If DEBUG Then
    <Assembly: AssemblyKeyFile("C:\PRIMAVERA.Public.snk")>
    <Assembly: AssemblyDelaySign(True)>
#Else
    <Assembly: AssemblyKeyName ("PRIMAVERA")>
    <Assembly: AssemblyDelaySign(False)>
#End If
```

These attributes define the assembly's signature and identity the manufacturing company. Each manufacturing company will create its own key file and substitute these values. What we need to do now, is creating the key file and register it on the development machine.

Create the following steps to create the key file and register it on the development machine:

- Open up the Visual Studio .NET command line prompt (**Start** | **All Programs** | **Microsoft Visual Studio 2008** | **Visual Studio Tools** | **Visual Studio 2008 Command Prompt**) and type in the following commands.

```
sn -k C:\<Company>.Public.Private.snk
sn -p C:\<Company>.Public.Private.snk C:\<Company>.Public.snk
```

Please substitute **<Company>** with the name of your company and keep in mind that the **AssemblyKeyFile** attribute in the **AssemblyInfo.vb** files in all projects must match this filename.

- Register the created key on the development machine by typing in following commands:

```
sn -tp C:\<Company>.Public.snk
sn -Vr *,XXXXXXXXXXXX
```

Where **XXXXXXXXXXXX** needs to be substituted by the value that resulted from the execution of the previous command.

Attention:

The described procedure only needs to be done on the development machine – however you need to remember that someday you'll compile a RELEASE version of the developed module and deploy it to the production server. In this case, you will need to copy the file **C:\<Company>.Public.Private.snk** to the production server and install the key using the following command:

sn -i C:\<Company>.Public.Private.snk <Company>

For more information about signing NET assemblies, please visit http://msdn.microsoft.com

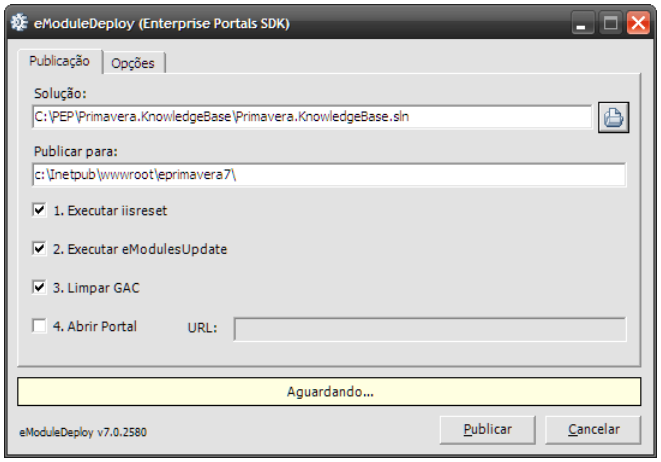
These files are automatically generated by Application Builder.

Step 1.3 – Publishing the new module

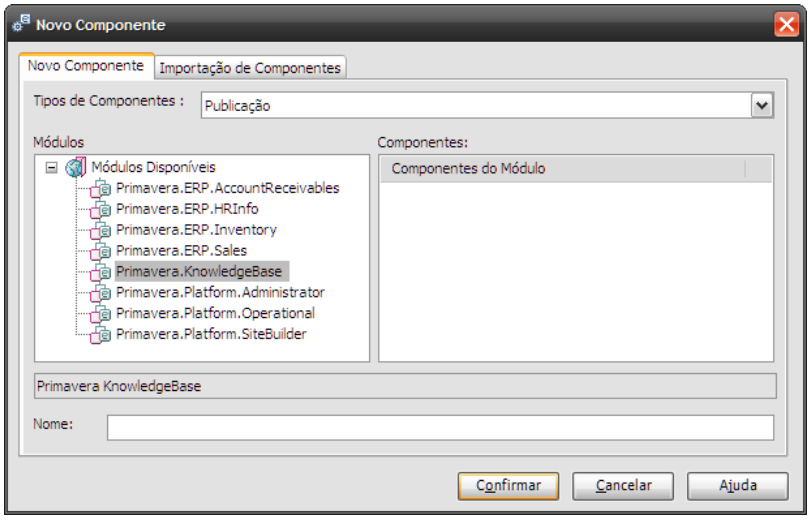
Now that we have created the base solution, defined the module's attributes and created the key file for signing the assemblies we are ready to test-drive the new module. The new module now should be compilable and deployable to the PRIMAVERA WebCentral Platform, even though it does not export any components yet. Anyway, we'd like to see the new module integrated, and check if everything is fine so far, so we proceed with the following steps:

- Use Visual Studio .NET to compile the whole solution in DEBUG mode.

- Close Visual Studio .NET and run the utility **eModuleDeploy**, which you can find in the operating system's start menu **Program Files** | **PRIMAVERA** | **WebCentral** | **SDK** | **eModuleDeploy**.
- Indicate the location of your module's Visual Studio solution
- It is recommended to activate always the options **Executar IISRESET** and **Executar eModulesUpdate** to guarantee the success of the operation. The option **Limpar GAC** should always be activated if you modified something in the WinUI project. Click on **Publicar** when you finished the configuration and to start the deployment process.



- After the module has been deployed, run the PRIMAVERA WebCentral Site Administrator and launch the Portal Designer to check that the new component is available, even though, it does not have any components yet.



Attention:

The deployment of a module will execute the following steps:

- Execute a restart of the Microsoft Internet Information Service (iisreset), which releases resources that might cause the deployment process to fail.

- Copy all module's assembly files, as well as *.aspx and *.ascx files to the PRIMAVERA's WebCentral installation folder (in our example, they will be copied to the subfolder **Modules/Primavera.KnowledgeBase**).
- Register the module information in the PRIMAVERA WebCentral database.
- Clean up the global assembly download cache – important for the WinUI assemblies which reside in this cache.

Step 2 – Creating a new component

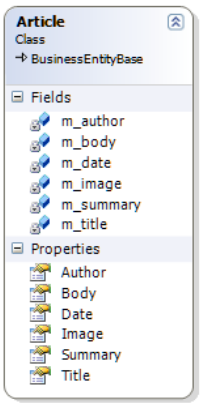
In this chapter we will create two new components for our module **Primavera.KnowledgeBase**. These two new components will be created as web user controls, one for listing all existing articles and one for viewing the details for a selected article.

Step 2.1 – Implementing the Business Entity

A business entity (BE) encapsulates all attributes of entity, in this case an article for our KnowledgeBase. A business entity only contains private attributes and public properties to gain access to the private attributes. This class will be utilized in all modules' project to store and transport information about an article of the KnowledgeBase.

- Add a new class named **Article** to the project **Primavera.KnowledgeBase.Entities**
- Implement the business entity class **Article** as shown in Appendix 1 [Page 110]

The class diagram on the right is the graphical representation of the class we just implemented. You should keep in mind that all business entities should inherit from the **BusinessEntityBase** class, which is provided by the PRIMAVERA WebCentral Platform. The PRIMAVERA WebCentral Platform also will take care of the business entities' serialization and deserialization to and from the database. The parameters of the **BusinessEntityAttribute** will specify name of the SQL Server table and a unique identifier for the business entity. The code fragment




```
<Serializable(), BusinessEntityAttribute("KNB_Articles", "…")> _
Public Class Article
    Inherits BusinessEntityBase
```

will tell the PRIMAVERA WebCentral Platform to serialize the entities data to a SQL table named **KNB_Articles**, which we will create later on in this chapter. Also, you already specified the property/table column mapping. The **BusinessEntityField** attribute, which you can find in front of a properties declaration, will tell the PRIMAVERA WebCentral Platform in which SQL Server column the data should be stored. Take for example the **Author** property:

```
<BusinessEntityField("Author")> _
Public Property Author() As String
    Get
        Return m_author
    End Get
    Set(ByVal Value As String)
```

```
            m_author = Value
        End Set
    End Property
```

The **BusinessEntityField** attribute's value **Author** will inform the PRIMAVERA WebCentral Platform to serialize the entities value to the **Author** column of the **KNB_Articles** table.

Step 2.2 – Implementing the Data Services

The Data Services project consists exclusively of classes that implement the data access to the SQL Server database and are called only by the Business Server layer. We'll implement all functionality that is needed to realize the data access, such as load, save and remove records from a table in the database. There isn't too much to do, though, as we inherit most of the functionality from the base class **DataServiceBase**.

- Add a new class called **Article** to the **Primavera.KnowledgeBase.Data** project.
- Implement the new class **Article** using the following code

```
Imports Primavera.Platform.Engine.Data
Imports Primavera.KnowledgeBase.Entities

<DataServiceAttribute(GetType(Entities.Article))> _
Public Class Articles
    Inherits DataServiceBase

End Class
```

As you see, there's not too much implementation to do here. All code that is necessary for the standard data access is already implemented in the base class **DataService**. However, if you need to customize the behaviour, you can use standard object oriented principles like overriding methods of the base class.

Step 2.3 – Implementing the Business Service

The business services (BS) implement the components business rules and serves as intermediate layer between user interface and data service layer. Here we will place code that is based on the data service layer and business entities layer. Its methods will be called from the two user interface layer – WebUI and WinUI.

- Add a new class called **Articles** to the **Primavera.KnowledgeBase.Business** project.
- Implement the business entity class **Article** as shown in Appendix 2 [Page 112].
- Analyse the code.

The next thing to do is to create a proxy class, which will provide a central access point to all business services objects. That way, the user interface code won't need to worry about declaring and instantiate an object from type **Article**. They simply call a static method which devolves a correctly instantiated **Article** object.

- Add a new class called **Proxy** to the **Primavera.KnowledgeBase.Business** project.

- Implement the new class **Proxy** using the following code

```
Public Class Proxy
    Inherits Primavera.Platform.Engine.Business.BusinessEngineBase

    Public ReadOnly Property ModuleID() As Guid
        Get
            Return New Guid("{5373dc00-2326-4ed6-b72f-aa5ead4cdcd1}")
        End Get
    End Property

    Public Shared Function Articles() As Business.Articles
        Return New Business.Articles
    End Function

End Class
```

Step 2.4 – Publishing the components to the Platform

The next modifications are necessary to publish the modules new components to the PRIMAVERA WebCentral Platform. The PRIMAVERA WebCentral Platform will call objects from the **Primavera.KnowledgeBase.Modules** project to retrieve information about the components this module provides. We will need to implement the necessary code for that the Platform recognizes the components we'll provide.

- Edit the file **ComponentsIds.vb**, which resides in the folder **Components** in the project **Primavera.KnowledgeBase.Modules**. As, later on, we will create two new components for the new module, we insert the implementation of two new properties:

```
    Public Shared ReadOnly Property KnowledgeBaseArticle() As Guid
        Get
            Return New Guid("{B6B669F2-35B2-43a1-A03D-7325EEB2B082}")
        End Get
    End Property

    Public Shared ReadOnly Property KnowledgeBaseArticleList() As Guid
        Get
            Return New Guid("{9763FE71-A201-42f6-BFF9-01E9A3449AE0}")
        End Get
    End Property
```

- Edit the file **Components.vb**, which also resides in the folder **Components** in the project **Primavera.KnowledgeBase.Modules**.
- Locate the method **List**. Here we will need to implement the code necessary to inform the PRIMAVERA WebCentral Platform about the existence of our 2 new components:

```
    Public Function List(ByVal ComponentType As ComponentType) As ServiceList
Implements IComponents.List

        Dim objCol As New ServiceList
        If (ComponentType = ComponentType.Administration) Then

        ElseIf ComponentType = ComponentType.Publishing Then
            objCol.Add(ComponentsIDs.KnowledgeBaseArticle, "Article Detail")
            objCol.Add(ComponentsIDs.KnowledgeBaseArticleList, "Article List")
        End If

        Return objCol

    End Function
```

- The **List** method's purpose is only to inform the PRIMAVERA WebCentral Platform about the existence of the 2 new components. Further details about each component will be introduced in the method **Edit**, which resides in the same file **Components.vb.** Modify this method the following way:

```
    Public Function Edit(ByVal WinContext As WinContext, ByVal ComponentID As
System.Guid) _
     As Component Implements IComponents.Edit

            Dim objComp As New Component
            If ComponentID.Equals(ComponentsIDs.KnowledgeBaseArticle) Then
                objComp.Name = "Article Detail"
                objComp.Description = "Component for showing a knowledge base article"
                objComp.ComponentClass = "Article"
                objComp.RequiresEnterprise = False
                objComp.IconIndex = 0
                objComp.BackgroundOpaque = True
                objComp.AllowsFrame = True
                objComp.ComponentSecurity = False

            ElseIf ComponentID.Equals(ComponentsIDs.KnowledgeBaseArticleList) Then
                objComp.Name = "Article List"
                objComp.Description = "Component for a list of articles"
                objComp.ComponentClass = "ArticleList"
                objComp.RequiresEnterprise = False
                objComp.IconIndex = 0
                objComp.BackgroundOpaque = True
                objComp.AllowsFrame = True
                objComp.ComponentSecurity = False

            End If
```

```
            objComp.ID = ComponentID
            Return objComp

        End Function
```

As you can imagine, the PRIMAVERA WebCentral Platform will call this method to initialize the components properties, whenever a component is placed on some webpage. In this method you will define if the component requires a connection to the Primavera ERP, if the security system should be supported, which icon should be displayed for the component, etc.

Attention:

Special attention while defining the value for the **ComponentClass** property of the component's object: The value of this property will determine the name of the web user control (ascx), which will be used by the PRIMAVERA WebCentral Platform to show the component. In this example, we will create web user controls named **Article.ascx** and **ArticleList.ascx**. The specified values for these properties need to match the names of the web user control class.

Step 2.5 – The component's user control

Until now, we have implemented the business entity, the business service, the data service layer as well as the code necessary for the PRIMAVERA WebCentral Platform to recognize the new component. Now we'll do the part that the actual user will see: The web-based user interface.

- In the project **Primavera.KnowledgeBase.WebUI**, create a new folder named **Components**. In this new folder, create a new web user control and name it **Article.ascx**. Change the designer view to HTML view and implement the web user control as shown in appendix 3 [page 115]
- Open the code behind file for the web user control **Article.ascx** and implement the code as shown in appendix 4 [page 116].
- Create another web user control in the same folder, named **ArticleList.ascx**. Change the designer to HTML and implement the web user control as shown in appendix 5 [page 118].
- Open the code behind file for the web user control **ArticleList.ascx** and implement the code as shown in appendix 6 [page 119]. Have special attention while implementing the method **GetDetailPageUrl**:

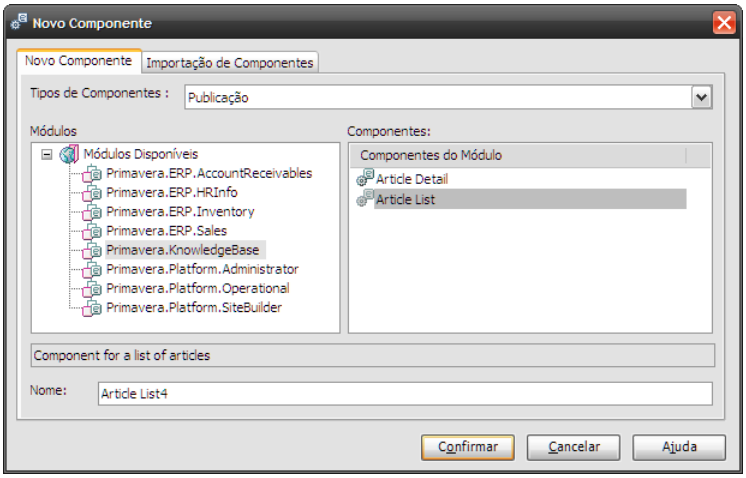
```
    Public Function GetDetailPageUrl(ByVal articleId As Guid) As String
        Dim guidCompArticleDetail As Guid = New Guid("B6B669F2-35B2-43a1-A03D-
7325EEB2B082")
        Return String.Format("ComponentRender.aspx?ComponentID={0}&ArticleID={1}",
guidCompArticleDetail, articleId)
    End Function
```

- The above method is called to generate an URL to navigate to the article detail component while browsing the article list. The static identifier specified for the local variable **guidCompArticleDetail** needs to match the identifier for the **Article** component. You can copy and paste this identifier from the file **ComponentsIds** in the **Primavera.KnowledgeBase.Modules** project.

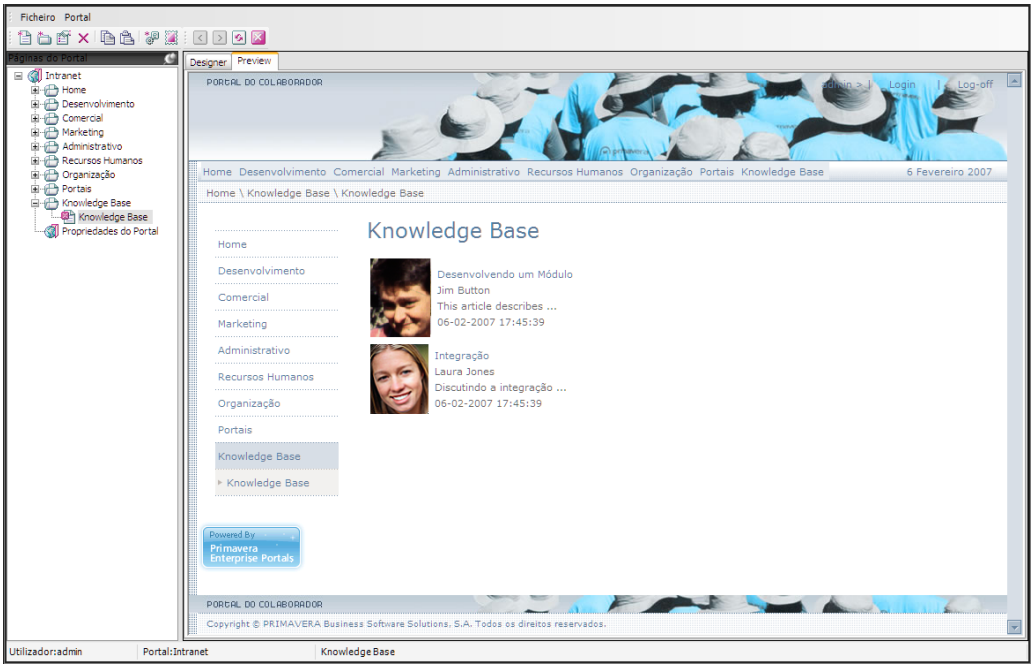
Step 2.6 – Compile, deploy and test the modifications made

To see the final result of the steps described in these last paragraphs we will need to compile the new module and deploy it to the PRIMAVERA WebCentral Platform.

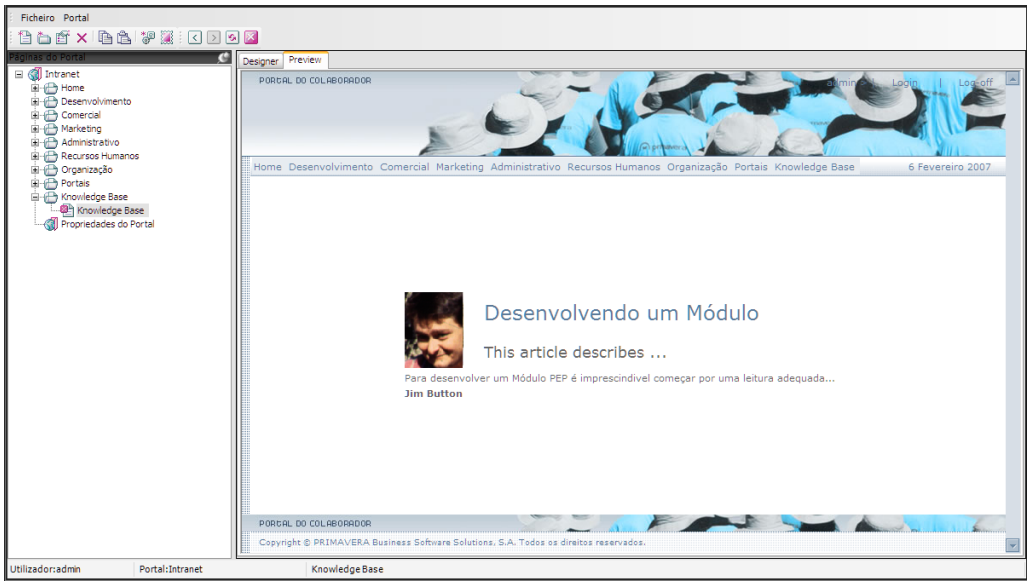
- Use the Visual Studio 2008 IDE to compile the solution in DEBUG mode.
- Run the **eModuleDeploy** utility, which you will find in the **Tools** folder of the PRIMAVERA WebCentral SDK.
- Indicate the complete filename of our module's solution and start the deployment process.
- The current solution already will try to read articles for the Knowledge Base from the database which was indicated while initializing the WebCentral Instance (**ePrimavera** by default). What we will need to do now, is creating the table structure and inserting some sample data to these tables. Please open up SQL Query Analyzer (or any other tool that permits the execution of SQL commands) and enter the SQL script you can find in Appendix 7 [page 120]. Execute this script on the WebCentral Database.
- Open up the Site Administrator application and navigate to the Portal Designer. Double-click on the **Intranet** portal. Create a new folder called **KnowledgeBase** and a new page called **Article List**. Place the new component **Article List** on this new page.



- Change to preview mode and you should see a list of two articles, which we created previously using the SQL scripts.



- Click on the link of one listed article and you should see the article's details.



Step 3 – The Article Administration Component

This chapter will show how to implement an administration component for the KnowledgeBase module. It will demand typing quite a lot code, both HTML as well as Visual Basic.NET, but soon you will see that this example introduces you to some important concepts and shows you how to reuse some of the controls that are provided by the PRIMAVERA WebCentral Platform.

Let's recapitulate the requirements for this control:

- A grid control will show all already existing articles

- It must be possible to insert new records as well as edit, remove and clone existing records
- It must be possible to edit all articles' properties, such as author, title, publishing date, image and the articles body.

All these requirements should be split into two user controls: One user control handles the **master grid view**, which shows all existing records, and one user control handles the **articles' details**. In fact, this way we can place the details user control into the master grid user control, and simply hide/ show the controls depending on the current mode. For example: If the administration control is in mode *consultation*, the master grid control will be shown and the details control will be hidden. If the user wishes to edit an existing record, the master grid control will be hidden, and the details control will be correctly initialized and shown to the user.

Step 3.1 – Publishing the Article Administration Component

The administration component, which we will try to implement in this chapter, is a regular component, which is published the same way we already did before. Just with one little difference. As you see in the component selection dialog, in the Portal Designer, PRIMAVERA WebCentral differentiate between Publishing and Administration components. We now are going to publish an administration component:

- Open the file **ComponentsIds.vb**, which you can find in the folder **Components** in the project **Primavera.KnowledgeBase.Modules**.
- Create a new shared, read-only property, which uniquely defines the new component:

```
    Public Shared ReadOnly Property KnowledgeBaseArticleAdmin() As Guid
        Get
            Return New Guid("{079DD2BA-F511-40a5-9360-2154DD8D6F27}")
        End Get
    End Property
```

- Open the file **Components.vb**, which also resides in the folder **Components** in the project **Primavera.KnowledgeBase.Modules**.
- Complete the method **List**, which we already edited before, to support the new administration component:

```
        Dim objCol As New ServiceList
        If (ComponentType = ComponentType.Administration) Then
            objCol.Add(ComponentsIDs.KnowledgeBaseArticleAdmin, "Article
Administration")

        ElseIf ComponentType = ComponentType.Publishing Then
            objCol.Add(ComponentsIDs.KnowledgeBaseArticle, "Article Detail")
            objCol.Add(ComponentsIDs.KnowledgeBaseArticleList, "Article List")
        End If
```

- We now need to implement the component's detailed attributes. The method **Edit** is the right place to do so. Insert the following code in the **ElseIf** path to the already existing **if** control path.

```
        ElseIf ComponentID.Equals(ComponentsIDs.KnowledgeBaseArticleAdmin) Then
            objComp.Name = "Article Administration"
```

```
            objComp.Description = "Component for Article Administration"
            objComp.ComponentClass = "ArticleAdmin"
            objComp.RequiresEnterprise = False
            objComp.IconIndex = 0
            objComp.BackgroundOpaque = True
            objComp.AllowsFrame = True
            objComp.ComponentSecurity = False
```

Now we already have exported the new component, which does not exist yet, but we're getting there…

Step 3.2 – Defining the Master Grid's Data Source

Before we begin with defining the user interface for the master grid control and the articles' detail control, let's define on what should appear in the master's data grid. The PRIMAVERA WebCentral Platform provides so called *table filters* for this purpose. Table Filters are implemented in the **Modules** project, and define which columns should appear in a data grid, as well as how they should be sorted and filtered. To implement a table filter:

- Open the file **TableFiltersIDs.vb**, which you can find in the folder **TableFilters** in the project **Primavera.KnowledgeBase.Modules**.
- Create a new shared, read-only property, which uniquely defines the new table filter:

```
    Public Shared ReadOnly Property ArticleTableFilter() As Guid
        Get
            Return New Guid("25662350-E994-45dd-9C75-340AA517357D")
        End Get
    End Property
```

- Open the file **TableFilters.vb**, which also resides in the folder **TableFilters** in the project **Primavera.KnowledgeBase.Modules**.
- We now need to implement the table filter's detailed attributes. This is done in the method **Edit**, which is called by the PRIMAVERA WebCentral Platform. Take a look at the following code segment and its code remarks:

```
    Public Function Edit(ByVal TableFilterID As Guid) As TableFilter Implements
ITableFilters.Edit

        Dim tableFilter As TableFilter = Nothing
        If TableFilterID.Equals(TableFiltersIDs.ArticleTableFilter) Then
            tableFilter = New TableFilter

            'define the master query and the field which will be used as primary key
            tableFilter.DomainQuery = "SELECT * FROM KNB_Articles"
            tableFilter.ReturnFields.Add("ID")

            'define a filter which configures the table filter grid
            Dim objPredefFilter As New PredefinedFilter()
            objPredefFilter.ID = New Guid("{41229DD9-0CC3-479f-A4BC-0B5A03EB7D90}")
            objPredefFilter.Name = "<All Articles>"
```

```
            'define the columns which will appear in the table filter grid
            objPredefFilter.Fields.Add("Author")
            objPredefFilter.Fields.Add("Date")
            objPredefFilter.WhereClause = ""
            objPredefFilter.FieldsOrder.Add("[Date] DESC")
            tableFilter.PredefinedFilters.Add(objPredefFilter)
            tableFilter.AllowCloning = True
        End If
        Return tableFilter
    End Function
```

- Implement the method like it is written above. We've now implemented a new table filter which will be bound to the master grid, showing the column Title, Author and Date and sorting all existing records descending by the column date.

Step 3.3 – Creating the Administration Detail User Control

The administration detail user control will be placed into the PRIMAVERA WebCentral Administration component. When a user firstly accesses the administration component, the administration detail user control will be hidden. It will only become visible if it is needed to show an existing items detail (edit) or when the user wishes to insert a new item (insertion).

The ASCX file does not need much explanation. Appendix 8 [Page 121] will show you the complete ASCX file content we'll need to complete this step. Just some labels, input controls and styles, to edit or insert an article from the KnowledgeBase. However, in the ASCX's code you will also find some extraordinary controls: The HTML editor as well as the image selection control. The web based HTML editor will permit to format the KnowledgeBase articles body. This is done through some JavaScript, which formats a plain textbox on the client side to a full-blown HTML editor. The image selection is done using two user controls from the PRIMAVERA WebCentral Platform: **WebImageSelectionPopUpButton** and **WebImageSelection**. The button serves to show the image selection user control and should be placed on the right side of the textbox which shows the relative path to the image selected. Whenever the user clicks on the image selection button, the KnowledgeBase item details are hidden and the image selection user control is shown.

- Add a new Web User Control, called **ArticleAdminDetail.ascx** to the folder **Components** in the **Primavera.KnowledgeBase.WebUI** project.
- Change the Visual Studio 2008 designer to the source view.
- Implement the ASCX file like shown in Appendix 8 [Page 121].

The code-behind file is more interesting, as it shows how to use PRIMAVERA WebCentral contextual information to load, edit and insert a record using the module's business service layer. We will now analyze the code-behind code step-by-step; you also can see the complete code-behind in Appendix 9 [Page 125].

- Open the user control's code behind file **ArticleAdminDetail.ascx.vb**
- As this user control not necessarily needs to be recognized directly by the PRIMAVERA WebCentral Platform, we won't need to change the classes' base class to **Platform.WebUI.WebComponentBase**, like we did before. This component embeds like normal ASP.NET user controls in other web pages or other user controls.

- We need to expose two events for the user control which will host the administration details. These events are triggered whenever the administration details user control needs to be hidden, say, when the user confirmed the modifications or he wants to cancel the operation. We declare these two events at the start of the classes implementation:

```
    'declare events to signal the parent user control or form that the user clicked
    'on the back button or the confirm button. the parent user control is responsible
    'for reacting the correct way on these events.
    Public Event BackButtonClicked()
    Public Event ArticleSaved()
```

- We placed some user controls in the ASCX file, which are not known to the Visual Studio IDE (as they reside in the PRIMAVERA WebCentral installation directory). By default, the Visual Studio IDE declares these user controls as **WebControls.UserControl** in the designer file, but as we need to access some of the controls properties and methods, it's a better way to declare these user controls with the correct type in the code-behind file:

```
    'declare user controls referenced from the PRIMAVERA WebCentral Platform
    'here in the code behind file, using the exact variable name as stated in the ascx
    Protected WithEvents WebImageSelectionPopUpButton1 _
     As Primavera.Platform.WebUI.WebImageSelectionPopUpButton
    Protected WithEvents controlImageSelection _
     As Primavera.Platform.WebUI.WebImageSelection
```

- We also need some private variables to store the contextual information, as well as the current edit mode (insertion or modification). These private attributes are only accessible using the public properties which we are going to implement next.

```
    'private attributes, which are initialized by the parent user control using the
    'public properties this class exposes
    Private _webContext As Primavera.Platform.WebUI.WebContext
    Private _visualizationMode As VisualizationMode
```

- The WebContext object contains essential information about the portals current context, such as which user is currently logged in, which organization, which template is being used as well as the database context, which we use to access the database. The Mode variable will be initialized by the parent form to let us now if we should insert a new record or edit an existing. It is good practise to avoid direct access to the classes attributes. These accesses should be realized using properties instead. We'll now implement the two properties to access the private members that we declared before:

```
    '<summary>
    '  This property allows access to vital context information, that are necessary to
    '  implement access to the system's database (such as information about user,
    '  organization, portal, etc). This property needs to be initialized from the
parent
    '  user control.
    '</summary>
    Public Property WebContext() As Primavera.Platform.WebUI.WebContext
        Get
```

```
            Return _webContext
        End Get
        Set(ByVal Value As Primavera.Platform.WebUI.WebContext)
            _webContext = Value
        End Set
    End Property

    '<summary>
    '  This property allows access to the edit mode; this control needs to know if we
are
    '  currently editing an existing, or inserting a new record. This property needs to
    '  be initialized from the parent user control.
    '</summary>
    Public Property Mode() As VisualizationMode
        Get
            Return _visualizationMode
        End Get
        Set(ByVal Value As VisualizationMode)
            _visualizationMode = Value
            Select Case _visualizationMode
                Case VisualizationMode.Insertion
                    CreateBusinessEntity()
                Case VisualizationMode.Edition
                    EditBusinessEntity()
            End Select
            ShowBusinessEntity()
        End Set
    End Property 'Mode
```

- Whenever the administration detail user control enters in edit mode, it will need to know the unique identifier of the article to edit. The parent form will initialize this property when the user clicks on the **Edit** button of the grid's toolbar.

```
    '<summary>
    '  The parent user control needs to initialize this property whenever the user is
    '  editing an existing record. This property needs to be initialized with the
    '  primary key of the record to be edited.
    '</summary>
    Public Property ArticleId() As Guid
        Get
            Return CType(Session("ArticleId"), System.Guid)
        End Get
        Set(ByVal Value As System.Guid)
            Session("ArticleId") = Value
        End Set
    End Property
```

- The following property will be used to persist an object of the business entity. As a web page itself is stateless, we will need to store it in the Session object. The Session object will be cleaned up by

the method **ClearSessionObjects**, whenever the user leaves the article detail administration form.

```
'<summary>
'  The following property persists an business entity object of the record that is
'  currently edited. This property is for internal use only.
'</summary>
Private Property BusinessEntity() As Entities.Article
    Get
        If Not Session("ArticleBEO") Is Nothing Then
            Return CType(Session("ArticleBEO"), Entities.Article)
        End If
        Return Nothing
    End Get
    Set(ByVal Value As Entities.Article)
        Session("ArticleBEO") = Value
    End Set
End Property
```

- As you saw in the ASCX file, there are some calls to functions in the code-behind to initialize the HTML editor. These functions are used to reference the textbox, which will be used as HTML editor as well as referencing the correct JavaScript file, which can be found on the web server. These helper function will be implemented as follows:

```
'<summary>
'  The following method will be called in the ASCX file to retrieve an HTML valid
'  identifier to reference the textbox control used as HTML editor (see also ASCX)
'</summary>
Protected Function ID2Name(ByVal ID As String) As String

    Return System.Text.RegularExpressions.Regex.Replace(ID, "(?<tok>[a-zA-Z0-
9]+)_", "${tok}:")

End Function 'ID2Name

'<summary>
'  The following method will be called to build a correct portal URL, for including
'  JavaScript files for the HTML editor (see also ASCX)
'</summary>
Protected Function GetCurrentPortalURL(ByVal Request As HttpRequest) As String

    Dim currentPortalUrl As String = String.Format("http://{0}/{1}", _
        Request.Url.Host, Request.Url.Segments(1))
    Return currentPortalUrl.TrimEnd("/")

End Function 'GetCurrentPortalURL
```

- The following method will be called whenever the user decides to confirm the current entity settings or to cancel the current operation and go back to the administration grid view. This method will remove the used objects which are stored in the session object.

```
'<summary>
'  The following method will cleanup the session object which are used by this
'  user control. You should call this method whenever the in the session state
'  persisted objects are no longer necessary.
'</summary>
Private Sub ClearSessionObjects()

    Session.Remove("ArticleId")
    Session.Remove("ArticleBEO")

End Sub 'ClearSessionObjects
```

- Next we need a method that is called whenever it is necessary to initialize the user control with the values currently stored in our business entity object. We also have to initialize the image selection control with the relative image path.

```
'<summary>
'  The following method will be called whenever it is necessary to initialize
'  the user control with values from the currently loaded business entity. This
'  is basically everytime the case, when a user edits an existing record.
'</summary>
Private Sub ShowBusinessEntity()

    txtAutor.Text = Me.BusinessEntity.Author
    txtArticle.Text = Me.BusinessEntity.Body
    txtDate.Value = Me.BusinessEntity.Date
    txtTitle.Text = Me.BusinessEntity.Title
    txtImage.Text = Me.BusinessEntity.Image

    'update image selection control
    WebImageSelectionPopUpButton1.ImageHtml = Me.BusinessEntity.Image
    WebImageSelectionPopUpButton1.SetConfigurationParameters()

End Sub 'ShowBusinessEntity
```

- The following method is called whenever the user needs to insert a new record to the KnowledgeBase articles. It will initialize a business entity object with default values.

```
'<summary>
'  The following method will be called whenever the user wishes to create
'  a new record. This method initializes a new business entity object.
'</summary>
Private Sub CreateBusinessEntity()

    'initialize new business entity
    Dim businessEntity As New Entities.Article
    businessEntity.ID = Guid.NewGuid()
    businessEntity.Author = WebContext.User.UserName
    businessEntity.Date = DateTime.Now
    businessEntity.Image = String.Empty
    businessEntity.Title = String.Empty
```

```
        businessEntity.Body = String.Empty

        'initialize image button
        WebImageSelectionPopUpButton1.ImageHtml = businessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

        Me.BusinessEntity = businessEntity

    End Sub 'CreateBusinessEntity
```

- Once the user clicked on the **Edit** button of the toolbar provided by the **WebTableFilterGrid** user control, the parent will initialize the details **ArticleId** property and set the mode to **Edition**. This is the place where the following method is called to load the business entity object with the values from database:

```
    '<summary>
    '  The following method will be called to initialize the internally used business
    '  entity object for being edited. It will load the specified record from database;
    '  the record's primary key needs to be set before.
    '</summary>
    Private Sub EditBusinessEntity()

        Dim businessLayer As New Business.Articles
        Me.BusinessEntity = businessLayer.Edit(WebContext.User.EngineContext,
Me.ArticleId)

        'initialize image button
        WebImageSelectionPopUpButton1.ImageHtml = Me.BusinessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'EditBusinessEntity
```

- Whenever the user clicked on the **Confirm** button, all modifications to the business entity need to be written to database. We do not need to care about if we are editing or inserting a record, because the business entity has an internal attribute that indicates if the object was loaded from database or not.

```
    '<summary>
    '  The following method will be called whenever it is necessary to write all
changes
    '  down to database. This method will update the existing record, which was edited,
or
    '  inserts a new record to the database.
    '</summary>
    Private Sub UpdateBusinessEntity()

        Me.BusinessEntity.Author = txtAutor.Text
        Me.BusinessEntity.Title = txtTitle.Text
        Me.BusinessEntity.Body = txtArticle.Text
        Me.BusinessEntity.Date = txtDate.Value
        Me.BusinessEntity.Image = txtImage.Text
```

```
        Dim businessLayer As New Business.Articles
        businessLayer.Update(WebContext.User.EngineContext, Me.BusinessEntity)

    End Sub 'UpdateBusinessEntity
```

- The **Page_Load** event handler method will be called whenever the client requests a web page that contains this user control – so that's a good place to do some initialization. The button that already handles the image selection treats the showing/ hiding of the panels, it just needs to know a reference to the panels controls.

```
    '<summary>
    '  The following event handler will be called whenever the page loads; place all
    '  initialization code here.
    '</summary>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Me.WebImageSelectionPopUpButton1.PanelMaster = Me.panelArticleDetails
        Me.WebImageSelectionPopUpButton1.PanelDetail = Me.panelImageSelection
        Me.WebImageSelectionPopUpButton1.WebImageSelectionControl =
Me.controlImageSelection
        If Not Me.IsPostBack Then
Me.WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'Page_Load
```

- The following event handler will be called whenever the user clicked on the **Back** button. In this case we will need to remove temporarily used objects from the Session object and raise the event that signals the parent form to hide the details panel and show the grid panel.

```
    '<summary>
    '  The following event handler will be called whenever the user clicked on the
"Back"
    '  button. No changes should be written down to database; the parent will receive
an
    '  event and should hide this details control and show the table filter grid.
    '</summary>
    Private Sub cmdBack_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnBack.Click

        RaiseEvent BackButtonClicked()
        ClearSessionObjects()

    End Sub 'cmdBack_Click
```

- The following event handler will be called whenever the user clicked the **Confirm** button. We will need to save all modifications on the business entity to database and raise the event that signals the parent form to hide the details panel and show the grid panel.

```
    '<summary>
    '  The following event handler will be called whenever the user clicked on the
"Confirm"
```

```
    '  button. All changes should be written down to database; ; the parent will
receive an
    '  event and should hide this details control and show the table filter grid.
    '</summary>
    Private Sub cmdSave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnConfirm.Click

        Me.UpdateBusinessEntity()
        RaiseEvent ArticleSaved()
        ClearSessionObjects()

    End Sub 'cmdSave_Click
```

- Whenever the user selected an image for the article, the following event handler will be raised. We will need to update the business entity as well as the textbox with the modified path to the selected image.

```
    '<summary>
    '  The following event handler will be called whenever the user confirmed an image
    '  in the image selection control. We will need to update the business entities
    '  image property with the new value.
    '</summary>
    Private Sub WebImageSelectionPopUpButton1_ImageSelected(ByVal configuration
As Platform.WebUI.ImageConfiguration) Handles
WebImageSelectionPopUpButton1.ImageSelected

        If Me.BusinessEntity Is Nothing Then _
            Exit Sub
        Me.BusinessEntity.Image = configuration.ImageHtml
        txtImage.Text = configuration.ImageHtml

    End Sub 'WebImageSelectionPopUpButton1_ImageSelected
```

That's all for the administration details user control, which supported editing and inserting new articles for the KnowledgeBase. As already said before, this user control will be placed on the main administration user control, which contains the list of all existing KnowledgeBase articles, as well as a toolbar that allows to edit/insert/remove and clone records. We also could have implemented the details user control's code directly to the main administration user control, but that would have been too much confusion to handle and maintain. In the next step we will implement the main administration user control.

Step 3.4 – Creating the Administration User Control

We already defined the table filter which we would like to use for our administration data grid as well as published our – until now – not existing component; now it's time to get our hands dirty and implement the administration user control. This user control will actually also encapsulate the articles' detail user control, and show/hide it in dependency of the current state. But let's start with the administration user control's HTML file:

- Add a new Web User Control, called **ArticleAdmin.ascx** to the folder **Components** in the **Primavera.KnowledgeBase.WebUI** project.
- Change the Visual Studio 2008 designer to the source view

- We will reuse various user controls that are provided by the PRIMAVERA WebCentral Platform; so let's first register the components we need (**WebGridMessage** and **WebGridTableFilter**) in the files header, using the **Register** tag: Below the **Control** tag, insert the following lines:

```
    <%@ Register TagPrefix="WebC" TagName="WebGridTableFilter"
Src="../Primavera.Platform/WebGridTableFilter.ascx" %>
    <%@ Register TagPrefix="WebC" TagName="WebGridMessage"
Src="../Primavera.Platform/WebGridMessage.ascx" %>
    <%@ Register Src="ArticleAdminDetail.ascx"
TagName="ArticleAdminDetail" TagPrefix="WebC" %>
```

- The next part of the ASCX file will be the panel that contains the data grid. We will encapsulate the data grid into a panel, so that we easily can show/hide it depending on the current edit mode.

```
    <!-- the following user control will be used to issue messages to the user -->
    <WebC:webgridmessage id="messageBox" runat="server"
visible="False"></WebC:webgridmessage>
    <asp:Panel ID="panelArticleMaster" Visible="True" runat=server>
        <!-- this panel will handle the grid that shows all available business entities
to
             the user. this "master" grid also implements common funcionalities like
insert
             new item, edit/clone and remove the current item. this panel will be
hidden by
             the code behind whenever the details (ArticleAdminDetail) needs to be
shown. -->
        <table id="tableArticles" runat="server" cellspacing="0" cellpadding="0"
width="100%" border="0">
         <tr>
             <td>
                 <WebC:webgridtablefilter id="articleAdminMasterGrid" visible="True"
                     runat="server" width="100%" height="20px" DoPagination="True"
                      AllowDefaultPaging="True" AllowDefaultSorting="True"
                      AllowSelectingRecordsPerPage="True"
DefaultPagingMode="NumericPages"
                      DefaultPagingPosition="Top" PageSize="10" PageSizeIndex="0">
                 </WebC:webgridtablefilter>
             </td>
         </tr>
        </table>
    </asp:Panel>
```

- Under this data grid panel, we will prepare another panel which will encapsulate the KnowledgeBase item details.

```
    <asp:Panel ID="panelArticleDetails" Visible="False" runat=server>
    <!-- this panel will handle the current entities details. it is shown whenever the
         user edits the currently selected or inserts a new record. we could have done
         this also in a separate aspx file. -->
```

```
<table>
<tr>
    <td>
        <WebC:ArticleAdminDetail id="articleAdminEntityDetails" runat="server">
        </WebC:ArticleAdminDetail></td>
    </td>
</tr>
</table>
</asp:Panel>
```

When finished, the user controls ASCX file should something look like the listing showed in Appendix 10 [Page 131]. Please compare your code with this listing and make sure everything's alright before we proceed with the code-behind file. You might get a warning issued from the Visual Studio IDE, telling you that the path in the **Register** tags, at the start of the file, are not valid. That's ok, because it already meets the correct directory structure in the production environment.

Now let's start with the code-behind file. The following steps will explain the sequence of changes that we are going to make in the code-behind file. If you have doubts on how it's done correctly, please refer to Appendix 11 [Page 133] where you can find the complete code-behind for this user control.

- Open the user control's code behind file **ArticleAdmin.ascx.vb**
- At the top of the file, import the namespaces that we are going to use in the code.

```
Imports Primavera.Platform.WebUI
Imports Primavera.Platform.WebUI.Controls
```

- The administration user control will encapsulate the list of existing records, as well as show the article details whenever the user is inserting a new or editing an existing record. The user control needs to persist the current editing mode (**Consultation** will show the data grid with existing records, **Insertion** and **Edition** will show the article detail user control). Therefore, we will need to declare a new enumeration type, which will be used by the administration user control, as well as the administration detail control. At the top of the file, between the Imports and the class declaration, insert following declaration:

```
Public Enum VisualizationMode
    Consultation
    Insertion
    Edition
End Enum
```

- As we are implementing a new component for the PRIMAVERA WebCentral Platform we need to change the user control's base class:

```
Partial Public Class ArticleAdmin
    Inherits Primavera.Platform.WebUI.WebComponentBase
```

- We placed some user controls in the ASCX file, which are not known to the Visual Studio IDE. By default, the Visual Studio IDE declares these user controls as **WebControls.UserControl** in the designer file, but as we need to access some of the controls properties and methods, it's a better way to declare these user controls with the correct type in the code-behind file:

```
'declare user controls referenced from the PRIMAVERA WebCentral Platform
```

```
    'here in the code behind file, using the exact variable name as stated in the ascx
    Protected WithEvents messageBox As Primavera.Platform.WebUI.Controls.WebGridMessage
    Protected WithEvents articleAdminMasterGrid As
Primavera.Platform.WebUI.Controls.WebGridTableFilter
```

- Now let's implement the property, which persists the user's control current visualization mode. As a user control by itself is stateless, we'll store the current mode into the session object. We will show/ hide the grid/ detail panel whenever the user control's visualization mode is changed:

```
    '<summary>
    '  This UserControl must handle various modes, as it joins the master and
    '  detail form. The following property will persist the current state, using the
    '  Session object. The initial state is consultation, which means, the master
    '  grid view is displayed. Whenever the user edits or inserts a new record, the
    '  master grid needs to be hidden and the details pane needs to be shown.
    '</summary>
    Private Property Mode() As VisualizationMode
        Get
            If Session("ArticleAdminMode") Is Nothing Then
                Return VisualizationMode.Consultation
            Else
                Return CType(Session("ArticleAdminMode"), VisualizationMode)
            End If
        End Get
        Set(ByVal Value As VisualizationMode)
            Select Case Value
                Case VisualizationMode.Consultation
                    panelArticleMaster.Visible = True
                    panelArticleDetails.Visible = False
                Case VisualizationMode.Insertion
                    panelArticleMaster.Visible = False
                    panelArticleDetails.Visible = True
                    articleAdminEntityDetails.Mode = VisualizationMode.Insertion
                Case VisualizationMode.Edition
                    panelArticleMaster.Visible = False
                    panelArticleDetails.Visible = True
                    articleAdminEntityDetails.Mode = VisualizationMode.Edition
            End Select
            Session("ArticleAdminMode") = Value
        End Set
    End Property 'Mode
```

- The following method is a helper method which will allow us to easily show a message box style information to the user. We will reuse the **WebGridMessage** user control, which is available through the PRIMAVERA WebCentral Platform.

```
    '<summary>
    '  The following method is used to show a message (or an error) to the user;
    '  it will show the platform's message box user control and initialize the
    '  icon/text to be shown.
    '</summary>
```

```
    Private Sub ShowMessage(ByVal Type As Platform.WebUI.Controls.WebGridMessageType,
ByVal Title As String, ByVal Message As String)

        messageBox.Visible = True
        messageBox.MessageType = Type
        messageBox.MessageTitle = Title
        messageBox.MessageText = Message
        messageBox.Refresh()

    End Sub 'ShowMessage
```

- The **Page_Load** event handler method will be called whenever the client requests a web page that contains the administration user control – so that's a good place to do some initialization. For example, the **WebGridTableFilter** user control needs to know which table filter it should use to display its data. The article detail user control needs to know contextual information for being able to load the business entity detail data from the database. While implementing this method, pay special attention to the **TableFilterID** property – this property needs to be initialized to unique identifier as declared before in the **TableFiltersIds** class.

```
    '<summary>
    '  The following method will be called when the page loads; it should be used
    '  to initialize the page's controls.
    '</summary>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        'initialize the table filter grid; we'll need to initialize this control with
module name
        'as well as table filter object (defined in the solutions Modules project),
which will be
        'used to query the information from the database.
        articleAdminMasterGrid.ModuleName = "Primavera.KnowledgeBase"
        articleAdminMasterGrid.TableFilterID = New Guid("{25662350-E994-45dd-9C75-
340AA517357D}")

        'initialize the details user control's web context property
        articleAdminEntityDetails.WebContext = Me.WebContext

        'if not postback: initialize the forms mode to its default (consultation)
        If Not Me.IsPostBack Then
            Mode = VisualizationMode.Consultation
            articleAdminMasterGrid.RefreshDataSource()
        End If
    End Sub 'Page_Load
```

- The **WebGridTableFilter** user control, by default, presents a toolbar which allows the user to easily insert, edit, remove and clone records. We now need to implement the code for these event handlers. Let's start with the event handler which will be called whenever the user decided to create a new record:

```
    '<summary>
```

```
    '  The following method will be called whenever the user clicked on the "New"
button
    '  of the table filter grid. The method will simply change the mode to insertion,
which
    '  will cause the table filter grid to be hidden and the details view to be shown.
    '</summary>
    Private Sub articleAdminMasterGrid_CreateRecord(ByVal Sender As Object)
Handles articleAdminMasterGrid.CreateRecord

        Me.Mode = VisualizationMode.Insertion

    End Sub 'articleAdminMasterGrid_CreateRecord
```

- Yes, that's all. We just set the current mode to **Insertion**. The property implementation will hide the administration grid and initialize the articles' details user control to insert a new record. The event handler, which is called to edit an existing record, is slightly more complex. It needs to initialize the article details **ArticleId** property, so it knows which article should be edited.

```
    '<summary>
    '  The following method will be called whenever the user clicked on the "Edit"
button
    '  of the table filter grid. The method will initialize the details view unique id
    '  property and set the edition mode, which will cause the table filter to be
hidden
    '  and the details view to be shown.
    '</summary>
    Private Sub articleAdminMasterGrid_EditRecord(ByVal Sender As Object,
ByVal selectedRowIdentifier As WebGridTableFilter.TableFilterKeyValueCollection,
ByRef cancel As Boolean) Handles articleAdminMasterGrid.EditRecord

        Try
            articleAdminEntityDetails.ArticleId = selectedRowIdentifier("ID")
            Mode = VisualizationMode.Edition
        Catch ex As System.Security.SecurityException
            Mode = VisualizationMode.Consultation
            ShowMessage(WebGridMessageType.Warning, String.Empty, ex.Message)
        Catch ex As Exception
            Mode = VisualizationMode.Consultation
            ShowMessage(WebGridMessageType.Error, String.Empty, ex.Message)
        End Try
    End Sub 'articleAdminMasterGrid_EditRecord
```

- The event handler for the **clone** operation won't need to change the visualization mode. It simply clones the currently selected record by using the module's business service layer, before it rebinds the grid with the data source.

```
    '<summary>
    '  The following method will be called whenever the user clicked on the "Clone"
button
```

```
    '  of the table filter grid. The code here demonstrates how to clone a existing
record.
    '</summary>
    Private Sub articleAdminMasterGrid_CloneRecord(ByVal Sender As Object,
ByVal selectedRowIdentifier As WebGridTableFilter.TableFilterKeyValueCollection,
ByRef cancel As Boolean) Handles articleAdminMasterGrid.CloneRecord

        Try
            Dim articleId As System.Guid = CType(selectedRowIdentifier("ID"),
System.Guid)
            Dim businessLayer As New Business.Articles
            businessLayer.Clone(WebContext.User.EngineContext, articleId)
            articleAdminMasterGrid.RefreshDataSource()

        Catch ex As System.Security.SecurityException
            ShowMessage(WebGridMessageType.Warning, String.Empty, ex.Message)

        Catch ex As Exception
            ShowMessage(WebGridMessageType.Error, String.Empty, ex.Message)

        End Try

    End Sub 'articleAdminMasterGrid_CloneRecord
```

- The **WebGridTableFilter** user control also exposes a toolbar button which should refresh the data shown in the grid - the event handler's implementation simply rebinds the grid to the data source:

```
    '<summary>
    '  The following method will be called whenever the user clicked on the "Refresh"
button.
    '  It will simply refresh the master grid and bind it to the data source.
    '</summary>
    Private Sub articleAdminMasterGrid_Refresh(ByRef Cancel As Boolean)
Handles articleAdminMasterGrid.Refresh

        articleAdminMasterGrid.RefreshDataSource()

    End Sub 'articleAdminMasterGrid_Refresh
```

- We also need to respond to events that are triggered by the article details user control. If the user clicked on the **Save** or the **Back** button, the administration user control needs to update the visualization mode and hide the article details control and update the records shown in the **WebGridTableFilter**.

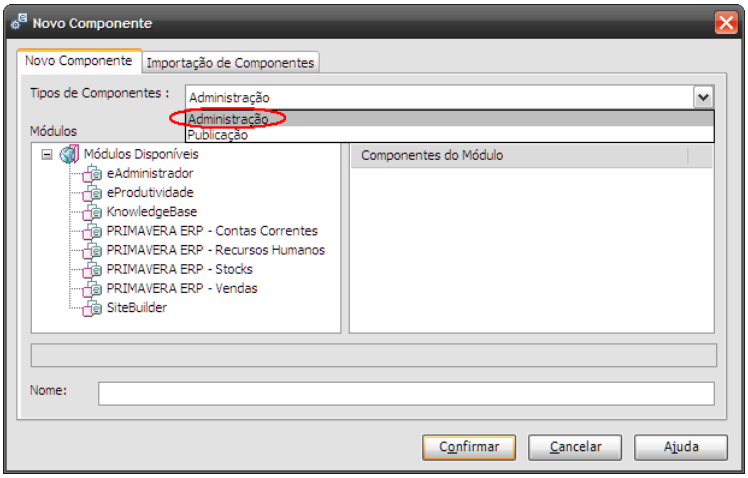
```
    '<summary>
    '  The following method will be called whenever the user clicked on the "Confirm"
button
    '  in the details view. This method sets the mode to consultation, which will cause
the
```

```
    '  table filter grid to be shown and the details control to be hidden.
    '</summary>
    Private Sub articleAdminEntityDetails_ArticleSaved()
Handles articleAdminEntityDetails.ArticleSaved

        Mode = VisualizationMode.Consultation
        articleAdminMasterGrid.RefreshDataSource()

    End Sub 'articleAdminEntityDetails_ArticleSaved

    '<summary>
    '  The following method will be called whenever the user clicked on the "Back"
button in
    '  the details view. This method simply sets the mode to consultation, which will
cause the
    '  table filter grid to be shown and the details control to be hidden.
    '</summary>
    Private Sub articleAdminEntityDetails_BackButtonClicked()
Handles articleAdminEntityDetails.BackButtonClicked

        Mode = VisualizationMode.Consultation

    End Sub 'articleAdminEntityDetails_BackButtonClicked
```

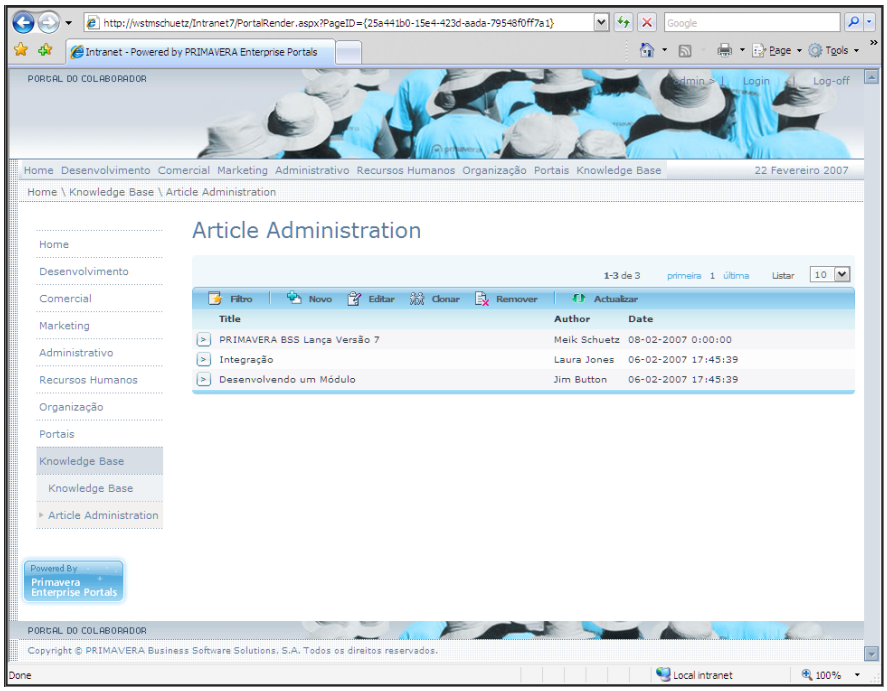
At this point, we should have done all modifications to support the administration of KnowledgeBase articles. We've got the grid, which will show us all existing articles and provides us with shortcuts to insert, edit, delete and clone KnowledgeBase articles. We implemented the article details view, which allows us to insert new –and edit existing KnowledgeBase articles. The solution should now be ready to compile without any errors. If you encounter an error during the compile process, please refer to the code implementation in the appendixes 8 to 11.

Step 3.5 – Testing the Article Administration Component

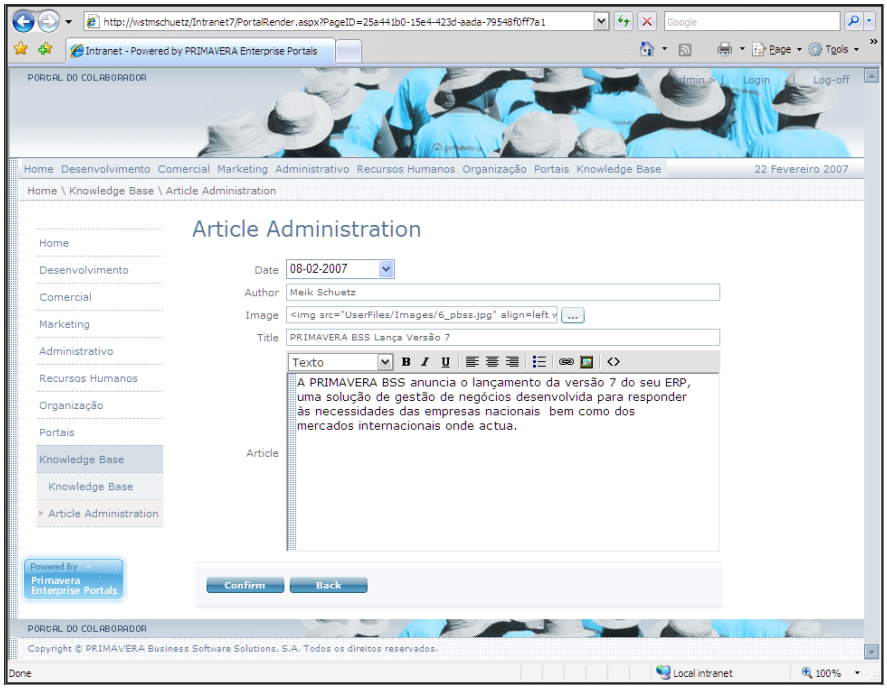
- Use the Visual Studio 2008 IDE to compile the solution in DEBUG mode.
- Close the Visual Studio 2008 IDE and run the **eModuleDeploy** utility, which you will find in the **Tools** folder of the PRIMAVERA WebCentral SDK.
- Indicate the complete filename of our module's solution and start the deployment process.
- After the deployment process has completed, launch the Site Administrator Application and navigate to the Portal Designer. Double-click on the **Intranet** portal. Create a new page **Article Administration** in the already existing folder **KnowledgeBase**.
- Right-click the page content panel and choose to place a new component on the page. In the component selection dialog, change the component type to **Administration** and select the **KnowledgeBase** module in the left panel.



- The **Article Administration** component should now appear in the list of available components. Select this component and click on **Confirm**.
- Check in the new page and publish the new portal revision.
- Open a new instance of your browser and navigate to the recently added page. You should see the new article administration component in action. The KnowledgeBase Article Grid should appear like in the image below.



- Try creating a new KnowledgeBase article and check editing the newly created article. The KnowledgeBase Article Detail component should be shown:



Step 4 – Component Customization

Imagine, whenever the user places the KnowledgeBase article list component on a page, you'd like to give the possibility to choose if the image should be shown in this list, or not – if the name of the author should appear, or not. Currently there's no way to configure the visualization attributes for any component that we implemented until now. This chapter will explain how to provide configuration properties to a component. In this chapter we'll implement a windows forms based configuration dialog, which will allow us to customize the article list component appearance.

Step 4.1 – Creating an entity to persist the configuration

First of all, we'll need a place to store the configuration information. The PRIMAVERA WebCentral Platform provides a concept named Property Bags, which allows us to store this kind of information as serialized object using XML. There's no need to create another SQL Server Table, as these Property Bags are stored in a common table in the WebCentral Database. We will need an entity which allows us to access this information, however.

- Open the **Primavera.KnowledgeBase.Entities** project and create a new folder named **PropertyBags**. With this folder selected, create a new class named **ArticleList**.
- Implement this class as shown in appendix 12 [page 138].

As you can see, there's nothing special about this classes implementation – just an entity class having private attributes and public properties to access them. The only thing to remember is to inherit from the base class **PropertyBagBase** and to give it a unique GUID value in the **SerializationEntity** attribute:

```
    <Serializable(), SerializationEntity("{A461C193-98FB-4a52-94AF-7EB6574B470E}",
"1.0")> _
    Public Class ArticleList
        Inherits PropertyBagBase
```

Step 4.2 – Configuration support in the business service layer

In this step we'll do the necessary alterations in the business service layer to permit the article list properties to be serialized to database.

- Open the **Primavera.KnowledgeBas.Business** project and create a new folder named **PropertyBags**. With this folder selected, create a new class named **ArticleList**.
- This new class is responsible for reading and updating the article list components properties. Implement the class like shown below.

```
    Imports Primavera.KnowledgeBase.Entities
    Imports Primavera.Platform.Engine.Entities
    Imports Primavera.Platform.Engine.Business

    Namespace PropertyBags

        <SerializationService(GetType(Entities.PropertyBags.ArticleList))> _
        Public Class ArticleList
            Inherits PropertyBagServiceBase

            Public Function Edit(ByVal Context As EngineContext, ByVal ID As
System.Guid) _
As Entities.PropertyBags.ArticleList
                Return BaseEdit(Context, ID)
            End Function

            Public Sub Update(ByVal Context As EngineContext, ByVal PropertyBag As
Entities.PropertyBags.ArticleList)
                BaseUpdate(Context, PropertyBag)
            End Sub
        End Class
    End Namespace
```

- Having the folder **PropertyBags** selected, create a new class named **Proxy**. This class will provide us with a central location to access property bags through its static properties.

```
    Namespace PropertyBags
        Public Class Proxy
            Public Shared ReadOnly Property ArticleList() As ArticleList
                Get
                    Return New ArticleList
                End Get
            End Property
        End Class
```

```
    End Namespace
```

Step 4.3 – Define the Interfaces for Remoting

The remoting layer – implemented through the assemblies IRemoting and Remoting – is necessary for the WinUI being able to access the business service layer. The remoting layer will marshal calls from the windows forms based client UI, which runs hosted by the Site Administration utility or your browser, to the business service layer which executes on the web server. Each call made from the client UI results in a HTTP request on the web server, which dispatches the request to the business service layer – so you should be very careful when it comes to remote calls as it greatly can influence your applications performance.

- Open the project **Primavera.KnowledgeBase.IRemoting** and create a new folder named **PropertyBags**. Having this folder selected, create a new class interface named **IArticles**.
- This interface will be used to marshal calls to access the article list components properties and only contains the signatures of the procedures, which are later implemented in the remoting project. Implement this interface class like shown below:

```
    Imports Primavera.Platform.Engine.IRemoting

    Namespace PropertyBags
        Public Interface IArticleList
            Function Edit(ByVal RemoteContext As RemoteEngineContext, _
ByVal ID As System.Guid) As Entities.PropertyBags.ArticleList
            Sub Update(ByVal RemoteContext As RemoteEngineContext, _
ByVal PropertyBag As Entities.PropertyBags.ArticleList)
            Function Exists(ByVal RemoteContext As RemoteEngineContext, _
ByVal ID As System.Guid) As System.Boolean
            Sub Remove(ByVal RemoteContext As RemoteEngineContext, ByVal ID As
System.Guid)
            Function Clone(ByVal RemoteContext As RemoteEngineContext, _
ByVal ID As System.Guid) As System.Guid
End Interface
    End Namespace
```

- In the **PropertyBag** folder, create a new class named **IProxy**, which will define the interface for the property bags proxy class. Like already implemented in the business service layer, the proxy class will provide a simple way to access all property bags a module exposes.

```
    Namespace PropertyBags
        Public Interface IProxy
            ReadOnly Property ArticleList() As IArticleList
        End Interface
    End Namespace
```

- In the project's root folder, create a new class named **IProxy**, which will define the interface for all remoting interfaces that the module supports. Currently we only use remoting for property bags, but we also could implement other windows based user interfaces. This proxy

class defines the global interface for all remoting classes implemented by the module. Implement this class like shown below.

```
Public Interface IProxy
    ReadOnly Property PropertyBags() As PropertyBags.IProxy
End Interface
```

Step 4.4 – Mapping the Remoting to the Business Layer

As already referred previously, the remoting layer channels calls from the client to the web server using NET remoting. To avoid coding the same functionality twice, the remoting layer simply maps calls to the Business Service Layer.

- Open the project **Primavera.KnowledgeBase.Remoting** and create a new folder named **PropertyBags**. Insert a new class **ArticleList** to that new folder.
- Implement the new class like shown in appendix 13 [page 140]

If you take a quick look at the code you'll see that the class **ArticleList** does inherit from the base class **RemoteServiceBase**, which is provided by the PRIMAVERA WebCentral Platform, and implements the interface contract **IArticleList**, which we previously defined. You also can see how each method of this class instantiates an object of the business layer and calls the appropriate method.

To easen up calls to the remoting layer, we also implement a proxy class, which will provide us an easy way to access the property bag from the windows form.

- Create a new class named **Proxy** in the **PropertyBag** folder of the project **Primavera.KnowledgeBase.Remoting**.
- Implement the new proxy class like shown below.

```
Namespace PropertyBags
    Public Class Proxy
        Inherits MarshalByRefObject
        Implements IRemoting.PropertyBags.IProxy

        Public ReadOnly Property ArticleList() As
IRemoting.PropertyBags.IArticleList _
            Implements IRemoting.PropertyBags.IProxy.ArticleList
            Get
                Return New ArticleList
            End Get
        End Property
    End Class
End Namespace
```

- Now we still need to implement the Proxy interface at module level. In the project's root folder, create a new class named **Proxy** and implement it like shown below.

```
Public Class Proxy
    Inherits Primavera.Platform.Engine.Remoting.RemoteProxyBase
```

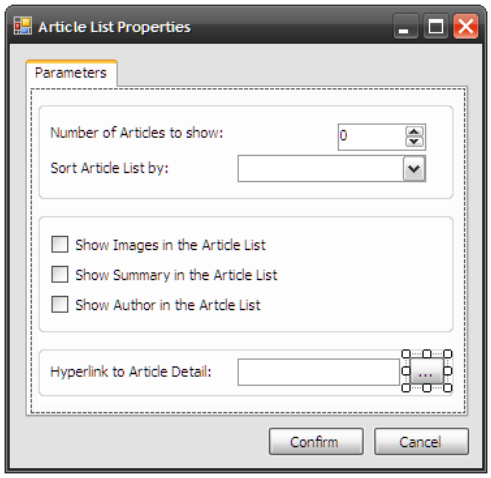
```
        Implements Primavera.KnowledgeBase.IRemoting.IProxy

        Public ReadOnly Property PropertyBags() As IRemoting.PropertyBags.IProxy _
            Implements IRemoting.IProxy.PropertyBags
            Get
                Return New Remoting.PropertyBags.Proxy
            End Get
        End Property
    End Class
```

Step 4.5 – Creating the Configuration Dialog

Now that we have implemented all necessary code for persisting the article list's configuration entity, we can start creating the configuration dialog in the WinUI project.

- Open the project **Primavera.KnowledgeBase.WinUI** and add a new reference to the assembly **Primavera.Platform.WinUI.dll**, which is located in the installation directory of the PRIMAVERA WebCentral SDK.
- In the WinUI project, create a new folder **PropertyBags** and insert a new Windows Form named **ArticleList**.

Now we should have an empty windows forms dialog where we will need to add some controls to the dialog, so that it looks more or less like the following screenshot:



The following table shows all controls that you should place from the Visual Studio IDE toolbox to the windows forms dialog:

Control	Name	Caption
Button	cmdConfirm	Confirm
Button	cmdCancel	Cancel
TabControl	tabParameters	Parameters
GroupBox	GroupBox1	
GroupBox	GroupBox2	
GroupBox	GroupBox3	

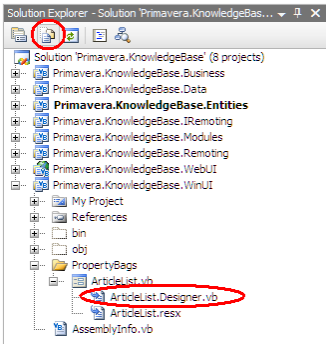
Label	lblArticleCount	Number of Articles to show:
NumericUpDown	txtArticleCount	
Label	lblArticleSort	Sort Article List by:
ComboBox	cmbArticleSort	
CheckBox	chkShowImage	Show Images in the Article List
CheckBox	chkShowSummary	Show Summary in the Article List
CheckBox	chkShowAuthor	Show Author in the Artcle List
Label	lblArticleDetailLink	Hyperlink to Article Detail:
TextBox	txtArticleDetailLink	
Button	cmdBrowseHyperLink	…

Place all these controls on the **ArticleList** Windows Forms dialog and format them, so that the dialog looks more or less like in the above screenshot.

- After all controls have been placed on the dialog, it is time to take a look at the code. The first thing to do is to import the necessary namespaces as well as to change the base class. Open the code behind file and declare the following namespaces at the beginning of the file.

```
Imports Primavera.Platform.Engine.IRemoting
Imports Primavera.KnowledgeBase.Entities
Imports Primavera.KnowledgeBase.IRemoting
```

- Next we will need to change the dialogs base class. This is done in file named **ArticleList.Designer.vb** which is normally hidden. You will need to click on the **Show All** toolbar button in the Solution Explorer to make it visible. The **ArticleList.Designer.vb** file is displayed as child of the ArticleList.vb file and contains all designer relevant code. Please change the class declarations base class so it matches the following code segment:




```
<Global.Microsoft.VisualBasic.CompilerServices.Designer
Generated()> _
    Partial Class ArticleList
        Inherits Primavera.Platform.WinUI.FormBase
```

- After changing the dialogs base class to **Primavera.Platform.WinUI.FormBase**, save and close the **ArticleList.Designer.vb** file as we won't need it anymore. In the **ArticleList.vb** file, add the following private attribute declaration into the ArticleList class.

```
    '<summary>
    '  The following private attribute will store the article lists current
    '  configuration settings.
    '</summary>
    Private articleListPropertyBag As Entities.PropertyBags.ArticleList
    Private componentLink As Platform.WinUI.ComponentLinkDialogOutput
```

- We also need a custom constructor method, which will receive a context for object initialization. This context variable, among other, contains information about the current user and the current portal. Implement the new constructor like this:

```
'<summary>
'  custom c'tor
'</summary>
Public Sub New(ByVal WinContext As WinContext)
    MyBase.New(WinContext)
    InitializeComponent()
End Sub 'New
```

- The dialog will be invoked whenever the user places a new ArticleList component on a page, or when the user wants to change the existing components properties. We need to provide two methods that will be called whenever the components properties need to be shown.

```
'<summary>
'  Call the following method to configure the properties for a new
'  component.
'</summary>
Public Function INew() As Guid
    InitializeForm()
    NewPropertyBag()
    ShowPropertyBag()
    If Me.ShowDialog() = DialogResult.OK Then
        Return articleListPropertyBag.ID
    Else
        Return Nothing
    End If
End Function 'INew

'<summary>
'  Call the following method to configure the properties for an already
'  existing component.
'</summary>
Public Sub IEdit(ByVal PropertyBagID As Guid)
    InitializeForm()
    EditPropertyBag(PropertyBagID)
    ShowPropertyBag()
    Me.ShowDialog()
End Sub 'IEdit
```

- The following method will initialize a new property bag instance for the ArticleList component with default values. It will be called whenever the user places a new ArticleList component on a page.

```
'<summary>
'  The following method is called to initialize a new property bag
'  entity, and is called whenever the user places a new component
'  on a page.
'</summary>
```

```
    Private Sub NewPropertyBag()
        articleListPropertyBag = New Entities.PropertyBags.ArticleList
        With articleListPropertyBag
            .ID = Guid.NewGuid
            .IncludeAuthor = True
            .IncludeImage = True
            .IncludeSummary = True
            .NumArticles = 5
            .Order = Entities.PropertyBags.ArticleListOrder.OrderByDate
            .DetailPageID = Guid.Empty
        End With
    End Sub
```

- The next two methods simply push data from the property bag object to the dialog's user interface and vice versa. These are two standard methods to synchronize the user interface and the proper bags object – nothing special about it. The only eye catcher here is the **ComponentLinkEdit** object. This is a dialog provided by the PRIMAVERA WebCentral Platform, which allows the user to easily select an existing page from any portal, and returns its full URL and unique page identifier. You can reuse this dialog wherever you think it's necessary.

```
    '<summary>
    '  The following method will be called to initialize the dialog's
    '  user interface controls with the property bag's values.
    '</summary>
    Private Sub ShowPropertyBag()
        With articleListPropertyBag
            txtArticleCount.Value = .NumArticles
            cmbArticleSort.SelectedIndex = .Order
            chkShowAuthor.Checked = .IncludeAuthor
            chkShowImage.Checked = .IncludeImage
            chkShowSummary.Checked = .IncludeSummary
            componentLink = Platform.WinUI.Dialogs.ComponentLinkEdit(WinContext,
.DetailPageID)
            If Not componentLink Is Nothing Then
                txtArticleDetailLink.Text = componentLink.PageName
            Else
                txtArticleDetailLink.Text = "<Default>"
            End If
        End With
    End Sub

    '<summary>
    '  The following method will be called to read the values in the
    '  dialog's user interface to to property bag object.
    '</summary>
    Private Sub ReadPropertyBag()
        With articleListPropertyBag
            .NumArticles = txtArticleCount.Value
            .Order = cmbArticleSort.SelectedIndex
```

```
            .IncludeAuthor = chkShowAuthor.Checked
            .IncludeImage = chkShowImage.Checked
            .IncludeSummary = chkShowSummary.Checked
            .DetailPageID = componentLink.PageID
        End With
    End Sub
```

- Of cause we need to synchronize the property bag object attributes with the WebCentral Database. The following methods will take care of reading and writing the property bags configuration from/to the WebCentral Database, using the remoting layer, which we implemented before.

```
    '<summary>
    '  The following method will be called to read an existing property
    '  bag from the WebCentral database.
    '</summary>
    Private Sub EditPropertyBag(ByVal PropertyBagID As Guid)
        Dim Context As Platform.Engine.IRemoting.RemoteEngineContext =
MyBase.CreateRemoteEngineContext()
        Dim Proxy As IRemoting.IProxy = Me.CreateRemoteProxy(GetType(IRemoting.IProxy))
        articleListPropertyBag = Proxy.PropertyBags.ArticleList.Edit(Context,
PropertyBagID)
    End Sub

    '<summary>
    '  The following method will be called to store the property bags
    '  attributes to the WebCentral database.
    '</summary>
    Private Sub UpdatePropertyBag()
        Dim Context As Platform.Engine.IRemoting.RemoteEngineContext =
MyBase.CreateRemoteEngineContext()
        Dim Proxy As IRemoting.IProxy = Me.CreateRemoteProxy(GetType(IRemoting.IProxy))
        Proxy.PropertyBags.ArticleList.Update(Context, articleListPropertyBag)
    End Sub
```

- We still did not implement the dialog control's initialization yet. The initialization method simply initializes the sorting order combo box items and sets the minimum and maximum number of records to show in the article list.

```
    '<summary>
    '  The following method initializes the dialogs controls
    '</summary>
    Private Sub InitializeForm()
        cmbArticleSort.Items.Clear()
        cmbArticleSort.Items.Add("Article Date")
        cmbArticleSort.Items.Add("Article Author")
        cmbArticleSort.Items.Add("Article Title")
        cmbArticleSort.SelectedIndex = 0

        txtArticleCount.Maximum = 15
        txtArticleCount.Minimum = 1
```

```
    End Sub
```

- But what should happen when the user clicks on the browse button to select a page for the article details? By default, WebCentral will use an empty page and automatically places there the article detail component in it, which shows the selected article details. Now, it is possible to create a custom page, just like any other page with other components, which will be called whenever the user clicks on an article in the list. The only requisite is that the page contains – among the other components – the **ArticleDetail** component. The following code will implement the page selection for the article detail page. Please make sure that the **ComponentID** property is equal to the component id for the **ArticleDetail** component, as specified in the file **ComponentsIds** in the **Primavera.KnowledgeBase.Modules** project.

```
    '<summary>
    '  The following method is called whenever the user clicked the Browse
    '  button, to select a page that contains the ArticleDetail component.
    '</summary>
    Private Sub btnLinkDetalhe_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) _
Handles cmdBrowseHyperlink.Click
        Dim dlgInput As New Platform.WinUI.ComponentLinkDialogInput()
        dlgInput.ComponentID = New Guid("{B6B669F2-35B2-43a1-A03D-7325EEB2B082}")
        dlgInput.PortalID = WinContext.Portal.PortalID
        dlgInput.PageID = articleListPropertyBag.DetailPageID
 Dim dlgOutput As Platform.WinUI.ComponentLinkDialogOutput = _
Primavera.Platform.WinUI.Dialogs.ComponentLinkSelect(WinContext, dlgInput)
        If Not dlgOutput Is Nothing Then
            articleListPropertyBag.DetailPageID = dlgOutput.PageID
            ShowPropertyBag()
        End If
    End Sub
```

- The only thing that's missing now, are the event handler methods for the Confirm –and Cancel buttons.

```
    '<summary>
    '  The following method is called whenever the user clicked the Confirm
    '  button. It will shift the configuration data from the user interface
    '  to the property bag object and store the changes in the Enterprise
    '  Portals database.
    '</summary>
    Private Sub btnConfirm_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) _
Handles cmdConfirm.Click
        ReadPropertyBag()
        UpdatePropertyBag()
        Me.DialogResult = DialogResult.OK
    End Sub

    '<summary>
    '  The following method is called whenever the user clicked the Cancel
```

```
    '  button. The user interface closes and no changes are done.
    '</summary>
    Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) _
Handles cmdCancel.Click
        Me.DialogResult = DialogResult.Cancel
    End Sub
```

After all these steps, the ArticleList dialog code behind file should somewhat look like stated in Appendix 14 [Page 142]. At this stage you can compile your project and see if there are any warnings or errors. If so, please verify the erroneous piece of code and compare it with the listing in Appendix 14.

Step 4.6 – Invoking the Configuration Dialog

All that is missing now is invoking the configuration dialog at the time the user places a new ArticleList component to a page, or whenever the user wishes to edit an existing components configuration. If you remember, the **Components** class in the project **Modules** provides two methods **NewPropertyPage** and **EditPropertyPage.** These two methods are called whenever it's necessary to show the configuration dialog. Change these two methods like shown below.

```
    Public Function NewPropertyPage(ByVal WinComponentContext As WinComponentContext)
As System.Guid _
Implements IComponents.NewPropertyPage
        If
WinComponentContext.ComponentID.Equals(ComponentsIDs.KnowledgeBaseArticleList) Then
            Dim dialog As New KnowledgeBase.WinUI.ArticleList(WinComponentContext)
            Return dialog.INew()
        Else
            Return Guid.NewGuid
        End If
    End Function

    Public Sub EditPropertyPage(ByVal WinComponentContext As WinComponentContext) _
Implements IComponents.EditPropertyPage
        If
WinComponentContext.ComponentID.Equals(ComponentsIDs.KnowledgeBaseArticleList) Then
            Dim dialog As New KnowledgeBase.WinUI.ArticleList(WinComponentContext)
            dialog.IEdit(WinComponentContext.PropertyBagID)
        End If
    End Sub
```

Step 4.7 – Applying the Configuration Settings

Once we implemented the possibility to define the configuration settings for the article list component, we'll also have to apply these configuration settings. Naturally this is done in the WebUI project, in the article list web user control.

- Open the project **Primavera.KnowledgeBase.WebUI** and open the code-behind file **ArticleList.ascx.vb**

- Add a new private member variable **propertyBag** to the class, which will store the configuration settings for the component.

```
    Private propertyBag As Entities.PropertyBags.ArticleList
```

- In the event handler **Page_Load**, insert a single line to load the configuration settings for the component.

```
    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        propertyBag =
ComponentContext.PropertyBag(GetType(Entities.PropertyBags.ArticleList))
        LoadArticles()
    End Sub
```

- Add a new function named **GetOrderByField**, which will return the name of the field, which should be used for sorting the grid's records.

```
    Private Function GetOrderByField() As String
        Select Case PropertyBag.Order
            Case Entities.PropertyBags.ArticleListOrder.OrderByAuthor
                Return "[author]"

            Case Entities.PropertyBags.ArticleListOrder.OrderByDate
                Return "[date]"

            Case Entities.PropertyBags.ArticleListOrder.OrderByTitle
                Return "[title]"

            Case Else
                Return "[date]"

        End Select
    End Function
```

- Add a new function named **BuildQuery**, which will return a SQL query based on the current component configuration.

```
    Private Function BuildQuery() As String
        Dim strSql As String = "SELECT TOP $1 * FROM KNB_Articles ORDER BY $2"
        Return Primavera.Platform.Engine.Business.SqlUtils.FormatSql(strSql, _
            propertyBag.NumArticles, GetOrderByField())
    End Function
```

- In the already existing method **LoadArticles**, replace the static SQL query with a call to the **BuildQuery** method.

```
    Private Sub LoadArticles()
        Dim strSql As String = BuildQuery()
        Dim tblArtigos As DataTable = Primavera.Platform.Engine.Business.List.List( _
WebContext.User.EngineContext, strSql)
        lstArtigos.DataSource = tblArtigos
```

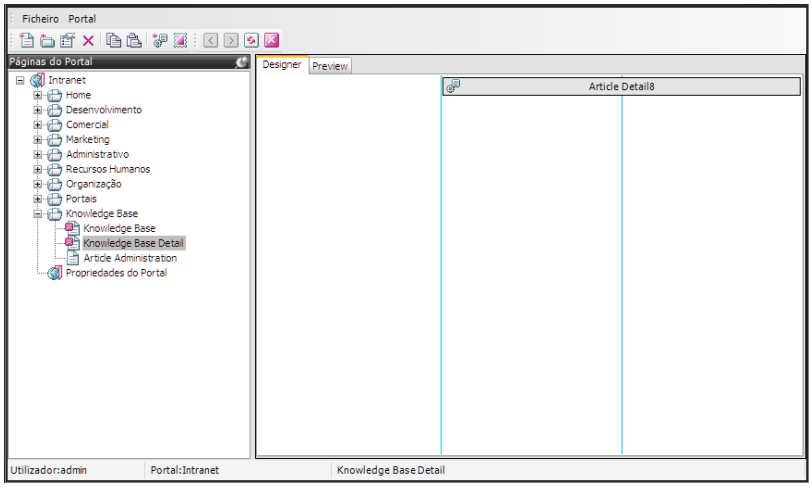
```
        lstArtigos.DataBind()
    End Sub
```

- We'll need to build a correct URL whenever the user specified a specific page to access the articles details. Modify the method **GetDetailPageUrl** so it matches following code (but make sure that the article detail guid matches the one you specified in the ComponentsIds.vb file).

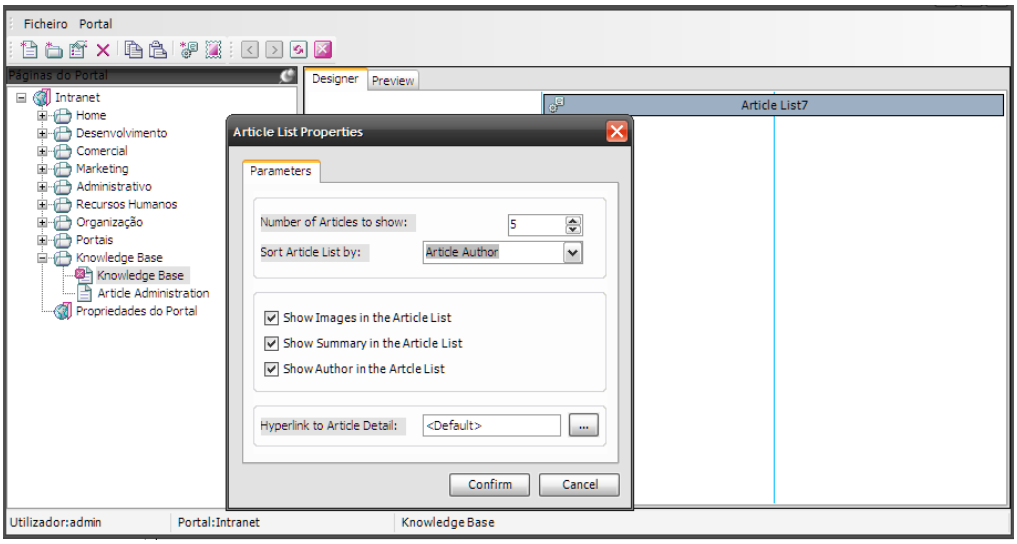
```
    Public Function GetDetailPageUrl(ByVal articleId As Guid) As String
        Dim guidCompArticleDetail As Guid = New Guid("B6B669F2-35B2-43a1-A03D-
7325EEB2B082")
        If PropertyBag.DetailPageId.Equals(Guid.Empty) Then
            Return String.Format("ComponentRender.aspx?ComponentID={0}&ArticleID={1}",
_
                guidCompArticleDetail, articleId)
        Else
            Return String.Format("PortalRender.aspx?PageID={0}&ArticleID={1}", _
                PropertyBag.DetailPageId.ToString(), articleId)
        End If
    End Function
```

Step 4.8 – Testing the Configuration Dialog

- Use the Visual Studio 2008 IDE to compile the solution in DEBUG mode.
- Close the Visual Studio 2008 IDE and run the **eModuleDeploy** utility, which you will find in the **Tools** folder of the PRIMAVERA WebCentral SDK.
- Indicate the complete filename of our module's solution and start the deployment process.
- After the deployment process has completed, launch the Site Administrator Application and navigate to the Portal Designer. Double-click on the **Intranet** portal.
- The portal should already have a folder **Knowledge Base**. Place a new page inside that folder and name it **Article Detail**. Place an article detail component on this page. This page will be our customized article detail page.



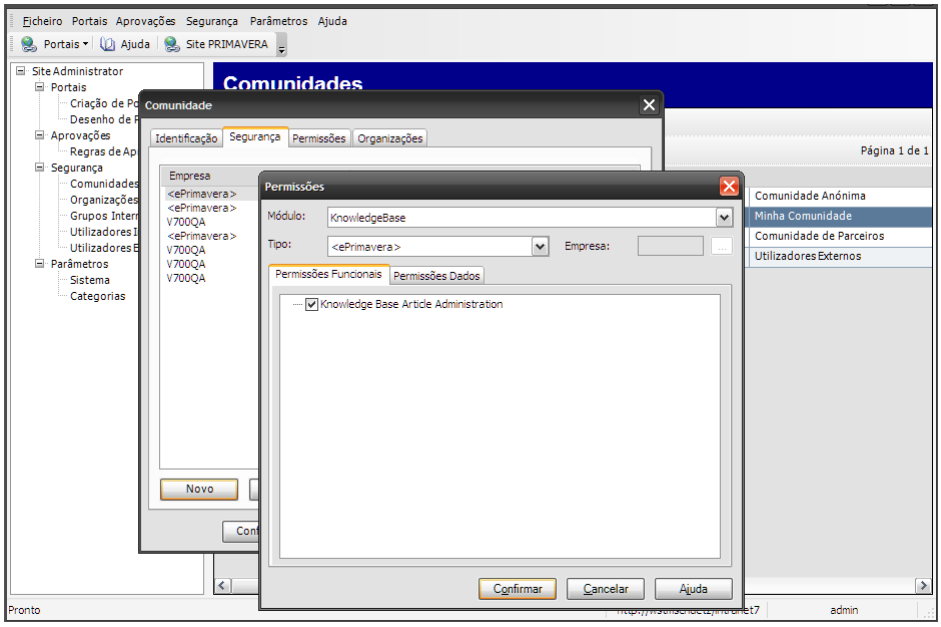
- Select the page **Knowledge Base** for editing. This page already should have an **Article List** component. We will need to remove this component from the page, as it does not have a configuration associated yet. Right-click the component and choose to remove the component.
- Place a new Article List component on the page. After confirming, it now should now open the configuration dialog for the component. Try changing the maximum number of articles to show in the list and/or changing the sort order. Try to specify the previously created page as new article detail page.



- The preview window should immediately reflect the changes made to the components configuration.

Step 5 – Component Security

One of the most important marks of PRIMAVERA WebCentral is its support for multi-level based security – the possibility to define permissions for each component, on a community, organization and user group level. In this chapter, we will explain how to integrate our customized component with the WebCentral's security system.



Step 5.1 – Support for the Component Security

The security support can be activated/ deactivated for each component. This option can be modified in the **Components** class in the project **Primavera.KnowledgeBase.Modules**.

- To enable the security support for the Article Administration component, open up the module **Primavera.KnowledgeBase.Modules** and edit the file **Components.vb**
- Locate the method **Edit** and the line in which the property **ComponentSecurity** is set to **False** for the Article Administration component. The fact that this property is set to **False** means that the component is always shown to any user, without any security validation.
- Change this value assignment so the property **ComponentSecurity** is set to **True**.

```
        ElseIf ComponentID.Equals(ComponentsIDs.KnowledgeBaseArticleAdmin) Then
            objComp.Name = "Article Administration"
            objComp.Description = "Component for Article Administration"
            objComp.ComponentClass = "ArticleAdmin"
            objComp.RequiresEnterprise = False
            objComp.IconIndex = 0
            objComp.BackgroundOpaque = True
            objComp.AllowsFrame = True
            objComp.ComponentSecurity = True

        End If
```

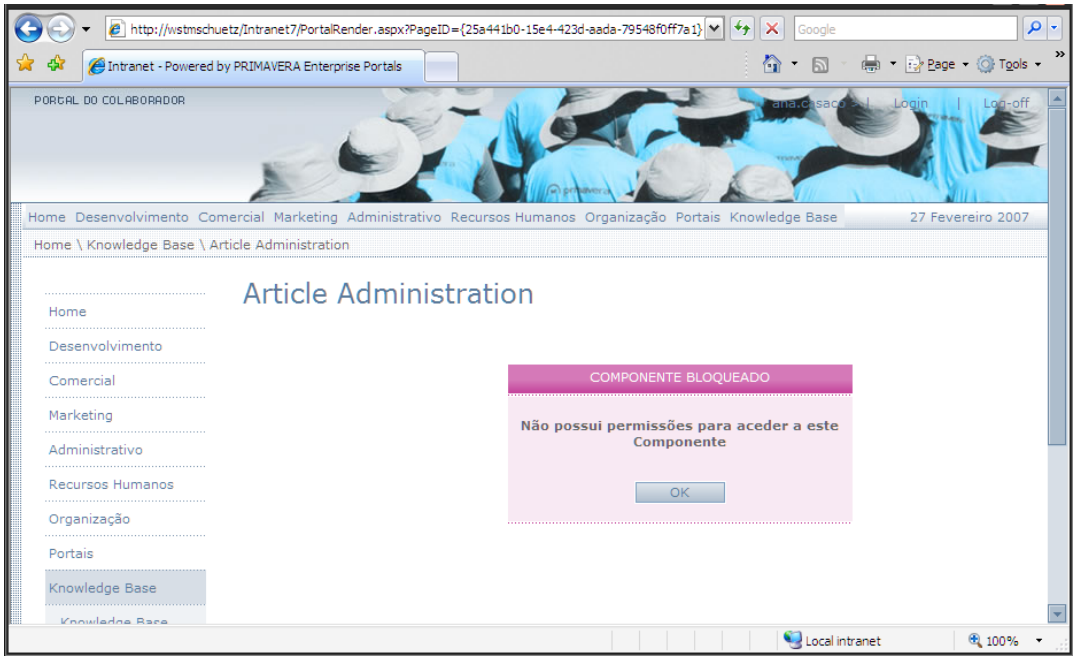
- To enable security support for the KnowledgeBase module we will need to build the permission tree. Still in the project **Primavera.KnowledgeBase.Modules**, open the class **Security** and locate the **method GetePrimaveraPermissionTree** and implement it like shown below.

```
        Dim objPermTree As New SecurityPermissionTree
        If PermissionTreeType = PermissionTreeType.Functional Then
            objPermTree.Add("ArticleAdmin", "Knowledge Base Article Administration", _
```

```
            ComponentsIDs.KnowledgeBaseArticleAdmin)

        ElseIf PermissionTreeType = PermissionTreeType.Data Then

        End If

        Return objPermTree
```

Step 5.2 – Testing the Component Security

- Use the Visual Studio 2008 IDE to compile the solution in DEBUG mode.
- Close the Visual Studio 2008 IDE and run the **eModuleDeploy** utility, which you will find in the **SDK** folder of the PRIMAVERA WebCentral.
- Indicate the complete filename of our module's solution and start the deployment process.
- After the deployment process has completed, launch the Site Administrator Application and navigate to the Portal Designer. Double-click on the **Intranet** portal.
- In the Portal Designer, make sure that you check-in all pages that are listed in the Knowledge Base folder. Publish the portal so that all modifications get online.
- Close the Portal Designer and navigate **Security** | **Communities**, which allows you to define the permissions for the community level. Double-click the community **Minha Comunidade**, select the security tab and add permission to access the KnowledgeBase Module.
- Confirm all changes and navigate to **Security | Organizations**, and add permissions to access the KnowledgeBase Module for the organization **Minha Organização**.
- Confirm all changes and navigate to **Security | Grupos Internos** and add permissions to access the KnowledgeBase Module for the usergroup **DID.**
- Try accessing the Article List Administration component using the portal Intranet and using the user **joao.fonseca**, which is member of the DID user group. The user has full access to the administration component.
- Now logoff and login using the user **ana.casaco**. Navigate to the Article List Administration component and you'll see that ana, which belongs to the group DRH, does not have sufficient permissions to see this component.



Step 6 – Supporting Multiple Languages

This exercise will explain how to implement multi language support to your components. We are going to prepare our KnowledgeBase project so it can accept and visualize the KnowledgeBase articles in various languages.

Step 6.1 – Preparing the business entities for localization

Currently, our Article business entity encapsulates all necessary attributes for a KnowledgeBase article in one single class. Supporting multiple languages means that we need to split these attributes in language dependent and independent attributes. There will be a master/detail relation, as there can be multiple language dependent records (details) for one single article (master). We'll need to remove the language dependent attributes from the master class **Article** and introduce a new (detail) class, named **ArticleCulture**, which holds the language dependent attributes (Article Title, Summary and Body).

- Create a new class named **ArticleCulture** – this class will hold all language dependent information and a reference to the master article (properties **ArticleId**, **CultureId**, **Title**, **Summary** and **Body**). See appendix 15 [page 146] for the complete code implementation.
- Create a new collection class **ArticleCultures** – this class will hold all available translations for a specific KnowledgeBase article. See appendix 16 [page 148] for the complete code implementation.
- In the project **PrimaveraKnowledgeBase.Entities**, open the class **Article** and remove the properties **Title**, **Summary** and **Body**.
- Still in the class **Article**, create a new private member attribute named **m_articleCultures,** as well as its public property accessors, from type **ArticleCultures**. This variable will store all article translations.

```
Private m_articleCultures As New ArticleCultures


<BusinessEntityDetailAttribute()> _
```

```
Public Property ArticleCultures() As ArticleCultures
    Get
        Return m_articleCultures
    End Get
    Set(ByVal Value As ArticleCultures)
        m_articleCultures = Value
    End Set
End Property
```

Step 6.2 – Changes in the Administration Component

We'll need to support the creation of articles in various languages; therefore we'll need to do some minor modifications to our KnowledgeBase article administration component. We'll simply provide the user a way to select a culture using an auto post back drop down list to the article details. Each time the user selects a new culture, the language dependent controls are respectively updated.

- Open the project Primavera.KnowledgeBase.WebUI and open the **ArticleAdminDetail.ascx** file for the article administration component.
- At the start of the HTML table that is used for data input, insert a new row which contains a label and a drop down, which will be filled with the available languages.

```
(omitted)
<table>
<tr>
    <!-- Culture Selection Box -->
    <td class="width_s alignRight">Culture</td>
    <td class="width_l">
        <asp:DropDownList ID="cultureSelection"
             runat="server" CssClass="data_input width_xl" AutoPostBack="True">
        </asp:DropDownList></td>
</tr>
<tr>
    <!-- Articles Publishing Date (Infragistics DateTime Picker) -->
    <td class="width_s alignRight">Date</td>
    <td class="width_s">
        <WebC:webdatechooser id="txtDate" runat="server"></WebC:webdatechooser>
    </td>
</tr>
(omitted)
```

- Switch to the code behind file and declare a new private member variable, which will be used to store the currently selected culture id.

```
'private attributes, which are initialized by the parent user control using the
'public properties this class exposes
Private _webContext As Primavera.Platform.WebUI.WebContext
Private _visualizationMode As VisualizationMode
Private m_activeCulture As Guid
```

- Insert a new private property, which will persist the language dependent data for the currently selected culture id.

```
'<summary>
'  The following property persists an language dependent business entity
'  object of the record that is currently edited. This property is for
'  internal use only.
'</summary>
Private Property BusinessEntityCulture() As Entities.ArticleCulture
    Get
        If Not Session("ArticleBEOCulture") Is Nothing Then
            Return CType(Session("ArticleBEOCulture"), Entities.ArticleCulture)
        End If
        Return Nothing
    End Get
    Set(ByVal Value As Entities.ArticleCulture)
        Session("ArticleBEOCulture") = Value
    End Set
End Property
```

- Locate the method **ClearSessionObjects** and make sure that the language dependent data stored in the session cache is properly removed. Add the following line to its implementation.

```
'<summary>
'  The following method will cleanup the session object which are used by this
'  user control. You should call this method whenever the in the session state
'  persisted objects are no longer necessary.
'</summary>
Private Sub ClearSessionObjects()

    Session.Remove("ArticleId")
    Session.Remove("ArticleBEO")
    Session.Remove("ArticleBEOCulture")

End Sub 'ClearSessionObjects
```

- Locate the method **ShowBusinessEntity**. This method makes references to already not existing properties of the entity object. We'll need to adapt the code so it shows the language dependent data. Change the following two property assignments.

```
'<summary>
'  The following method will be called whenever it is necessary to initialize
'  the user control with values from the currently loaded business entity. This
'  is basically everytime the case, when a user edits an existing record.
'</summary>
Private Sub ShowBusinessEntity()

    txtAutor.Text = Me.BusinessEntity.Author
    txtArticle.Text = Me.BusinessEntityCulture.Body
```

```
        txtDate.Value = Me.BusinessEntity.Date
        txtTitle.Text = Me.BusinessEntityCulture.Title
        txtImage.Text = Me.BusinessEntity.Image

        'update image selection control
        WebImageSelectionPopUpButton1.ImageHtml = Me.BusinessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'ShowBusinessEntity
```

- Locate the method **CreateBusinessEntity** in which we'll need to update the code, so that an entity for the default culture is being correctly created and initialized.

```
    '<summary>
    '  The following method will be called whenever the user wishes to create
    '  a new record. This method initializes a new business entity object.
    '</summary>
    Private Sub CreateBusinessEntity()

        'initialize new business entity
        Dim businessEntity As New Entities.Article
        businessEntity.ID = Guid.NewGuid()
        businessEntity.Author = WebContext.User.UserName
        businessEntity.Date = DateTime.Now
        businessEntity.Image = String.Empty

        Dim businessEntityCulture As New Entities.ArticleCulture
        businessEntityCulture.ArticleID = businessEntity.ID
        businessEntityCulture.CultureID = m_activeCulture
        businessEntity.ArticleCultures.Add(businessEntityCulture)

        'initialize image button
        WebImageSelectionPopUpButton1.ImageHtml = businessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

        Me.BusinessEntity = businessEntity
        Me.BusinessEntityCulture = businessEntityCulture

    End Sub 'CreateBusinessEntity
```

- Locate the method **EditBusinessEntity**, which also makes references to not existing properties in the business entity object. We'll need to assure that the object that stores the language dependent attributes is correctly initialized.

```
    '<summary>
    '  The following method will be called to initialize the internally used business
    '  entity object for being edited. It will load the specified record from database;
    '  the record's primary key needs to be set before.
    '</summary>
```

```
    Private Sub EditBusinessEntity()

        Dim businessLayer As New Business.Articles
        Me.BusinessEntity = businessLayer.Edit(WebContext.User.EngineContext,
Me.ArticleId)
        Me.BusinessEntityCulture =
Me.BusinessEntity.ArticleCultures.GetItemByCultureID(m_activeCulture)

        If Me.BusinessEntityCulture Is Nothing Then
            Me.BusinessEntityCulture = New Entities.ArticleCulture
            Me.BusinessEntityCulture.ArticleID = BusinessEntity.ID
            Me.BusinessEntityCulture.CultureID = m_activeCulture
            Me.BusinessEntity.ArticleCultures.Add(BusinessEntityCulture)
        End If

        'initialize image button
        WebImageSelectionPopUpButton1.ImageHtml = Me.BusinessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'EditBusinessEntity
```

- Navigate to the method **UpdateBusinessEntity** which is called whenever the component needs to write down the changes made to an article to database. We'll need to serialize the current values from the user interface to the article object.

```
    '<summary>
    '  The following method will be called whenever it is necessary to write all
changes
    '  down to database. This method will update the existing record, which was edited,
or
    '  inserts a new record to the database.
    '</summary>
    Private Sub UpdateBusinessEntity()

        Me.BusinessEntity.Author = txtAutor.Text
        Me.BusinessEntity.Date = txtDate.Value
        Me.BusinessEntity.Image = txtImage.Text

        Me.BusinessEntityCulture.Title = txtTitle.Text
        Me.BusinessEntityCulture.Body = txtArticle.Text

        Dim businessLayer As New Business.Articles
        businessLayer.Update(WebContext.User.EngineContext, Me.BusinessEntity)

    End Sub 'UpdateBusinessEntity
```

- Now let's implement the method which will be responsible for populating the drop down list with the available cultures. The WebCentral Platform provides a method named **ListInstalledLanguages**, which returns a table of all available languages. We'll use this

method to bind it against the drop down list's data source.

```
    '<summary>
    '  The following method will be called to bind the installed portal cultures
    '  with the dropdown that allows the user to select a culture.
    '</summary>
    Public Sub InitializeCultureSelection()

        If Not Page.IsPostBack() Then
            Dim proxy As New Primavera.Platform.Security.Business.Parameters
            cultureSelection.DataSource =
proxy.ListInstalledLanguages(WebContext.User.EngineContext)
            cultureSelection.DataTextField = "Culture"
            cultureSelection.DataValueField = "ID"
            cultureSelection.SelectedIndex = 0
            cultureSelection.DataBind()
        End If
        If (m_activeCulture.Equals(Guid.Empty)) Then _
            m_activeCulture = New Guid(cultureSelection.SelectedValue)

    End Sub 'InitializePortalCultures
```

- Call the new method **InitializeCultureSelection** from the **Page_Load** event handler.

```
    '<summary>
    '  The following event handler will be called whenever the page loads; place all
    '  initialization code here.
    '</summary>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        'initialize culture selection
        InitializeCultureSelection()

        Me.WebImageSelectionPopUpButton1.PanelMaster = Me.panelArticleDetails
        Me.WebImageSelectionPopUpButton1.PanelDetail = Me.panelImageSelection
        Me.WebImageSelectionPopUpButton1.WebImageSelectionControl =
Me.controlImageSelection
        If Not Me.IsPostBack Then
Me.WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'Page_Load
```

- Implement a new event handler called whenever the user changes the currently selected culture in the web user control. This code needs to assure that the previous language dependent attributes are correctly stored and show the language dependent values for the new selected culture.

```
    '<summary>
    '</summary>
    Private Sub cultureSelection_SelectedIndexChanged(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles cultureSelection.SelectedIndexChanged

        'store the updated values for the current culture
        Me.BusinessEntityCulture.Title = txtTitle.Text
        Me.BusinessEntityCulture.Body = txtArticle.Text

        'change the current culture and show the culture dependent values
        m_activeCulture = New Guid(cultureSelection.SelectedValue)
        Me.BusinessEntityCulture =
Me.BusinessEntity.ArticleCultures.GetItemByCultureID(m_activeCulture)
        If Me.BusinessEntityCulture Is Nothing Then
            Me.BusinessEntityCulture = New Entities.ArticleCulture()
            Me.BusinessEntityCulture.ArticleID = Me.BusinessEntity.ID
            Me.BusinessEntityCulture.CultureID = m_activeCulture
            Me.BusinessEntity.ArticleCultures.Add(Me.BusinessEntityCulture)
        End If
        txtTitle.Text = Me.BusinessEntityCulture.Title
        txtArticle.Text = Me.BusinessEntityCulture.Body

    End Sub 'cultureSelection_SelectedIndexChanged
```

Step 6.3 – Changes in the Visualization Component

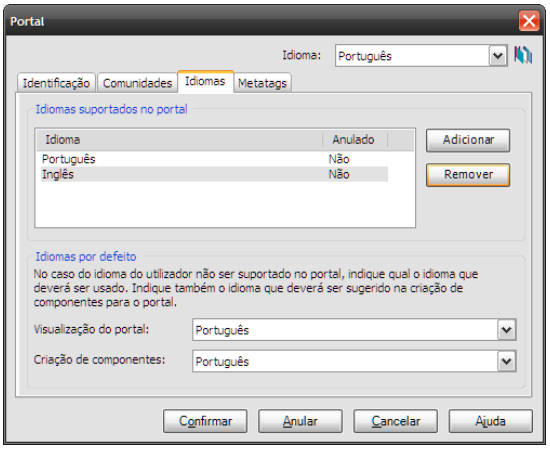
Now let's see what needs to be done in the visualization components. The user logged in has a language associated, but also can change the default language using the country flags that appear in the portal header. The **WebContent.User** object has significant information about the current user and can be used to determine the current language for the user session. With this information we now can easily perform the changes in the article list component, as we just need to update the SQL query to reflect the currently selected culture.

- In the WebUI project, Open the file **ArticleList.ascx.vb** and locate the method **BuildQuery**. Replace it's implementation so that it meets the following code segment:

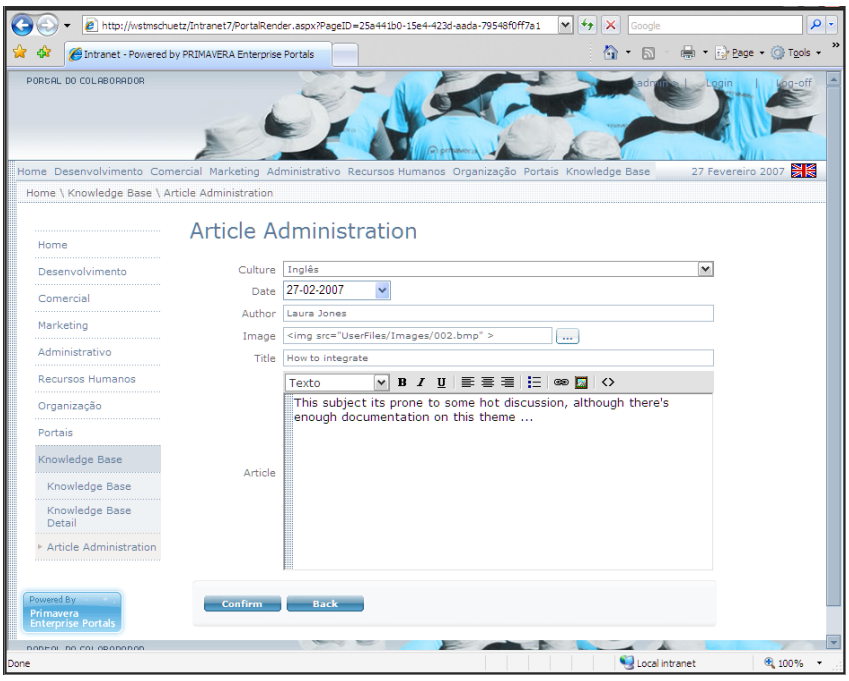
```
    Private Function BuildQuery() As String
        Dim strSql As String = "SELECT TOP $1 * FROM KNB_Articles "
        strSql &= vbCrLf & "INNER JOIN KNB_ArticleCultures ON "
        strSql &= vbCrLf & "  KNB_ArticleCultures.ArticleID = KNB_Articles.ID"
        strSql &= vbCrLf & "  AND KNB_ArticleCultures.CultureID = @2 ORDER BY $3"
        Return Platform.Engine.Business.SqlUtils.FormatSql(strSql, _
            propertyBag.NumArticles, _
            WebContext.User.Culture.CultureID, _
            GetOrderByField())
    End Function
```

Step 6.4 – Testing the multi-language support

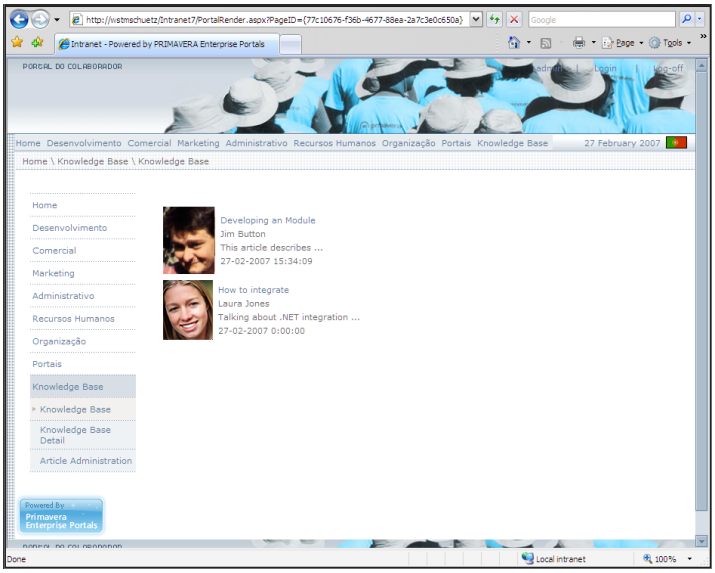
- Use the Visual Studio 2008 IDE to compile the solution in DEBUG mode.
- Close the Visual Studio 2008 IDE and run the **eModuleDeploy** utility, which you will find in the **Tools** folder of the PRIMAVERA WebCentral SDK.
- Indicate the complete filename of our module's solution and start the deployment process.
- You might have noticed that we changed the database table structure for the multi-language support. Please use the SQL Query Analyzer and the script you can find in appendix 17 [page 150] to update the database structure.
- Currently, the portal **Intranet** just supports one single language. We will need to define this portal as multi language. Therefore, launch the Site Administration utility and navigate to **Portais | Criação de Portais**. Double-click the record for the Intranet portal and select the Language tab. Add support for the English language to the Intranet portal and confirm.



- Open a browser and navigate to the KnowledgeBase Administration page. Edit one of the existing articles and try to edit the article in English.



- Try to change the current language using the drop down listbox at the top of the form. The language dependent values should be updated correctly.
- Confirm the modified article and navigate to the KnowledgeBase Article List. The list appears in the current default language.
- Click on the english flag in the portal header. The KnowledgeBase Article List should now appear in english.



Step 7 – Supporting Approvals

The WebCentral Platform for Approvals permits the definition of workflows and approval states. The PRIMAVERA WebCentral uses this approval platform in some human resources components as well as in the content management components, like press releases, messages, etc.

This exercise will explain how our customized KnowledgeBase component can take advantage of the approval platform. We'll also dive into content categories. Categories are used to differentiate content of the same type. See, for example, *internal* messages and *public* messages. Categories also permit to define permissions on a different level than component based. Having the category model implemented, you will be able to define permissions per content category type, e.g. user xpto does not have permissions to view internal messages.

Step 7.1 – Preparing the Business Entities for Approvals

First of all, we'll need some new private attributes and public accessor properties in the business entity class. These new attributes will store information about the current approval state as well as the associated categories for a given KnowledgeBase article.

- Open the project **Primavera.KnowledgeBase.Entities** and edit the file **Article.vb** that contains our business entity class declaration. Insert the following variable declarations to the list of private member attributes:

```
Private m_article As String
Private m_published As Boolean
Private m_categories As Primavera.Platform.Services.Entities.EntityCategories = _
    New Primavera.Platform.Services.Entities.EntityCategories
```

```
    Private m_locked As Boolean
    Private m_approvalId As Guid
```

➢ Add the new public accessor properties to the KnowledgeBase Article Entity:

```
    <BusinessEntityField("Article")> _
    Public Property Article() As String
        Get
            Return m_article
        End Get
        Set(ByVal Value As String)
            m_article = Value
        End Set
    End Property

    <BusinessEntityField("Published", IsReadOnly:=True)> _
    Public Property Published() As Boolean
        Get
            Return m_published
        End Get
        Set(ByVal Value As Boolean)
            m_published = Value
        End Set
    End Property

    <BusinessEntityDetailAttribute(ForeignField:="EntityID")> _
    Public Property Categories() As
Primavera.Platform.Services.Entities.EntityCategories
        Get
            Return m_categories
        End Get
        Set(ByVal Value As Primavera.Platform.Services.Entities.EntityCategories)
            m_categories = Value
        End Set
    End Property

    <BusinessEntityField("Locked", IsReadOnly:=True)> _
    Public Property Locked() As Boolean
        Get
            Return m_locked
        End Get
        Set(ByVal Value As Boolean)
            m_locked = Value
        End Set
    End Property

    <BusinessEntityField("ApprovalID", IsReadOnly:=True)> _
    Public Property ApprovalID() As Guid
        Get
            Return m_approvalId
```

```
        End Get
        Set(ByVal Value As Guid)
            m_approvalId = Value
        End Set
    End Property
```

- Open the file **ArticleList.vb** in the folder **PropertyBags** and declare a new private member variable as well as a new public accessor property:

```
        Private m_colCategories As New GuidCollection
        Public Property Categories() As GuidCollection
            Get
                Return m_colCategories
            End Get
            Set(ByVal Value As GuidCollection)
                m_colCategories = Value
            End Set
        End Property
```

Step 7.2 – Preparing the Data Services for Approvals

Currently, our Data Service Layer does not implement any methods – in fact it inherits all standard operations from its base class **DataServiceBase**. We now need to insert two customized methods to the data service layer, which are responsible for querying approval information from the database.

- Open the file **Article.vb** from the project **Primavera.KnowledgeBase.Data** and insert the following two methods to the class.

```
    '<summary>
    '  The following method will query the approval rule which was associated to
    '  to the specified categories.
    '</summary>
    Public Function GetApprovalRuleID( _
        ByVal objContext As Primavera.Platform.Engine.Entities.EngineContext, _
        ByVal objCategories As Primavera.Platform.Engine.Entities.Fields) As Guid

        Dim strSQL As String = "SELECT TOP 1 [ApprovalRuleID] FROM [PLT_Categories] " _
            & "WHERE [ID] IN $1 AND [ApprovalRuleID] IS NOT NULL"
        Dim strCats As String = String.Empty
        Dim objField As Primavera.Platform.Engine.Entities.Field

        For Each objField In objCategories
            strCats = strCats + SqlUtils.FormatSql("@1, ", CType(objField.Value, Guid))
        Next
        If (strCats.Length > 0) Then
            strCats = "(" + strCats.Substring(0, strCats.Length - 2) + ")"
        End If
        strSQL = SqlUtils.FormatSql(strSQL, strCats)
```

```
        Dim objCmd As New System.Data.SqlClient.SqlCommand(strSQL,
objContext.RequestConnection)
        Dim guidApprovalRuleID As Guid = CType(objCmd.ExecuteScalar(), Guid)
        objContext.ReleaseConnection()
        objCmd.Dispose()

        Return guidApprovalRuleID

    End Function 'GetApprovalRuleID

    '<summary>
    '  The following method will return identifier and title of a knowledge
    '  base article for a given approval instance identifier.
    '</summary>
    Public Sub GetContentFromRuleID( _
        ByVal objContext As Primavera.Platform.Engine.Entities.EngineContext, _
        ByVal guidApprovalId As Guid, _
        ByRef guidArticleId As Guid, _
        ByRef strTitle As String)

        Dim strSQL As String = SqlUtils.FormatSql("SELECT [ID],[Article] " _
            & "FROM [KNB_Articles] WHERE [ApprovalID] = @1", guidApprovalId)
        Dim objTable As DataTable = List.List(objContext, strSQL)
        If (objTable.Rows.Count = 1) Then
            guidArticleId = CType(objTable.Rows(0)("ID"), Guid)
            strTitle = CType(objTable.Rows(0)("Article"), String)
        End If
    End Sub
End Class
```

Step 7.3 – Preparing the Business Layer for Approvals

- In the **Primavera.KnowledgeBase.Business** project, open the file **Articles.vb** and implement a new method which will verify the article publishing permission for the current user. This method will simply call a method of its base class to check either the current user has permission to publish a message for the associated article categories.

```
    Public Overloads Function VerifyPublishingPermission(ByVal Context As _
        EngineContext, ByVal ID As Guid, ByRef OutMessage As String) As Boolean
        Return BaseVerifyPublishingPermission(Context, ID, OutMessage)
    End Function
```

- Approval rules will be invoked whenever a user tries to *publish* an article. Currently, our administration component does not support the publishing concept – an article is automatically published whenever the user *saves* it. In order to support the publishing concept, we'll need to add some methods to the **Article** class:

```
    Public Sub Publish(ByVal Context As EngineContext, ByVal ID As Guid)
        BasePublish(Context, ID)
    End Sub
```

```
    Public Sub UnPublish(ByVal Context As EngineContext, ByVal ID As Guid)
        BaseUnPublish(Context, ID)
    End Sub

    Public Overloads Sub Lock(ByVal objContext As EngineContext, _
ByVal guidID As Guid, ByVal guidApprovalID As Guid)
        BaseLock(objContext, guidID, guidApprovalID)
    End Sub

    Public Overloads Sub UnLock(ByVal objContext As EngineContext, _
ByVal guidID As System.Guid)
        BaseUnLock(objContext, guidID)
    End Sub
```

- Implement the two methods which are necessary to support categories for this module. Still in the **Article** class, add the following two methods:

```
    Public Sub MoveCategoryItems(ByVal Context As EngineContext, ByVal SourceCategoryID
As Guid, _
        ByVal DestCategoryID As Guid)
        BaseMoveCategoryItems(Context, SourceCategoryID, DestCategoryID)
    End Sub

    Public Sub RemoveCategoryItems(ByVal Context As EngineContext, ByVal CategoryID As
Guid)
        BaseRemoveCategoryItems(Context, CategoryID)
    End Sub
```

- Our business layer still needs to wrap the calls to the two methods that we introduced to the data layer to support the approvals. Insert the following two methods to the **Article** class.

```
    Public Function GetApprovalRuleID(ByVal objContext As EngineContext, _
        ByVal objCategories As Primavera.Platform.Engine.Entities.Fields) As Guid

        Dim objDSO As New Data.Articles
        Return objDSO.GetApprovalRuleID(objContext, objCategories)
    End Function

    Public Sub GetContentFromRuleID(ByVal objContext As EngineContext, _
        ByVal guidApprovalID As Guid, ByRef guidArticleID As Guid, ByRef strTitle As
String)

        Dim objDSO As New Data.Articles
        objDSO.GetContentFromRuleID(objContext, guidApprovalID, guidArticleID,
strTitle)
    End Sub
```

Step 7.4 – Implementing Approvals in the Module

- Open the project **Primavera.KnowledgeBase.Modules** and add a reference to the assembly **Primavera.Platform.Services.Business**, which can be found in the PRIMAVERA WebCentral SDK **bin** folder.
- In the **Approvals** folder, edit the file **ApprovalTypesIDs.vb**. This class contains unique approval identifiers, which are used by the PRIMAVERA WebCentral Platform to distinguish between the various types of approvals. Each content type which is subject to approval must have a unique identifier.

```
Public Shared ReadOnly Property Articles() As Guid
    Get
        Return New Guid("{448E2B4B-979E-4c1a-878E-769BD0D7FEFD}")
    End Get
End Property
```

- Open the file **ApprovalInstances.vb** for editing. This class contains all the logic for the article approval, which are quite a few lines of code. Please refer to appendix 18 [page 152] to see the necessary implementation.
- Please make sure that the identifier **MyModuleID**, which is declared at the top of the file, matches the identifier for your module (you can consult this identifier in the resource file **Res.resx** for the project **Primavera.KnowledgeBase.Modules**).
- The file **ApprovalRules.vb** has the approval rule configuration for this module. Localize the method **ApprovalRuleConfig** and implement the following lines:

```
Public Function ApprovalRuleConfig(ByVal guidTypeID As System.Guid) _
    As ApprovalRuleConfig Implements IApprovalRules.ApprovalRuleConfig

    'In this module all content types use the same configuration
    Dim objRuleConfig As New ApprovalRuleConfig
    With objRuleConfig
        .ID = guidTypeID 'Unique id for each content type
        .StartStateBlocked = True
        .EndStatesBlocked = True
        .IntermediateStatesBlocked = False
        .IntermediateStatesAllowed = True

        .StartState.ID = New Guid("{FB596820-9BF3-45b8-B0E3-3EE9A7A16DC8}")
        .StartState.Name = "Pending"

        .EndApprovedState.ID = New Guid("{67999B91-17E1-4cdb-A161-539767A8C408}")
        .EndApprovedState.Name = "Published"

        .EndRejectedState.ID = New Guid("{86B398B6-5386-4001-B324-5E4DE03CD9CF}")
        .EndRejectedState.Name = "Rejected"

        .AllowNotifications = True
        .SimpleApprovalsOnly = False

    End With
```

```
        Return objRuleConfig

    End Function
```

- Now we just need to enable approvals for our modules – this is done in the **Approvals** class. Localize the methods **SupportRules** and **SupportInstances** and change their return values from **False** to **True**.
- The method **ListTypes** returns a collection of supported content approvals. Here we need to make sure that our new approval type is correctly registered and returned to the caller.

```
    Public Function ListTypes() As ServiceList Implements IApprovals.ListTypes

        Dim objCol As New ServiceList
        objCol.Add(ApprovalTypesIDs.Articles, "Article Categories")
        Return objCol

    End Function
```

Step 7.5 – Implementing Categories in the Module

- Open the file **CategoryTypeIds.vb**, which you can find in the folder **Categories** of the project **Primavera.KnowledgeBase.Modules**. Let's define a new identifier for the category of KnowledgeBase articles.

```
    Public Shared ReadOnly Property Articles() As Guid
        Get
            Return New Guid("{B9F475F4-2C54-463f-8FD9-F6318EDAE971}")
        End Get
    End Property
```

- The class **Category** will provide further information on the module's supported categories. Locate the method **List** and modify the code so that it returns the new category as result.

```
    Public Function List() As ServiceList Implements ICategories.List

        Dim objCol As New ServiceList
        objCol.Add(CategoryTypesIDs.Articles, "Categorias de Artigos")
        Return objCol

    End Function
```

- Next we need to specify the categories details, which takes place in the **Edit** method. Here we also will let the PRIMAVERA WebCentral Platform know that the new category supports the approval workflow.

```
    Public Function Edit(ByVal CategoryTypeID As System.Guid) As Category Implements
ICategories.Edit

        Dim objCat As New Category
```

```
        If CategoryTypeID.Equals(CategoryTypesIDs.Articles) Then
            objCat.Name = "Categorias de Artigos"
            objCat.SupportsApprovals = True
            objCat.ApprovalTypeID = ApprovalTypesIDs.Articles
        Else
            Return Nothing          ' Item not found
        End If

        objCat.ID = CategoryTypeID
        Return objCat

    End Function
```

Step 7.6 – Updating the Article Administration UI

The user interface that we currently have for introducing the KnowledgeBase Articles is a bit inappropriate for the new functionality that we are implementing – we need to be able to attach categories to the article as well as support the publishing mechanism. In this step we'll polish the current user interface to support these new features.

- First of all, we will have to make sure that only published articles appear in the list of KnowledgeBase Articles. Open the project **Primavera.KnowledgeBase.WebUI** and edit the file **ArticleList.ascx.vb**. Locate the **BuildQuery** method and add an additional SQL WHERE statement to the query that is returned by the method:

```
    Private Function BuildQuery() As String
        Dim strSql As String = "SELECT TOP $1 * FROM KNB_Articles "
        strSql &= vbCrLf & "INNER JOIN KNB_ArticleCultures ON "
        strSql &= vbCrLf & "  KNB_ArticleCultures.ArticleID = KNB_Articles.ID"
        strSql &= vbCrLf & "  AND KNB_ArticleCultures.CultureID = @2 " _
            & "WHERE KNB_Articles.Published = 1 ORDER BY $3"
        Return Platform.Engine.Business.SqlUtils.FormatSql(strSql, _
            propertyBag.NumArticles, _
            WebContext.User.Culture.CultureID, _
            GetOrderByField())
    End Function
```

- Now let's get to the administration user interface. Open the **ArticleAdminDetail.ascx** file and register three new user controls, provided by the PRIMAVERA WebCentral Platform, which will allow us to select categories for the KnowledgeBase Articles and display information messages.

```
    <%@ Register TagPrefix="WebC" TagName="WebCategoryPopUpButton"
     Src="../Primavera.Platform/WebCategoryPopUpButton.ascx" %>
    <%@ Register TagPrefix="WebC" TagName="WebCategorySelection"
     Src="../Primavera.Platform/WebCategorySelection.ascx" %>
    <%@ Register TagPrefix="WebC" TagName="WebGridMessage"
Src="../Primavera.Platform/WebGridMessage.ascx" %>
```

- Place the message box control right before we declare the article detail panel – this message box will be used to display information about the current state of the KnowledgeBase Article we are editing.

```
    (ommitted)
    <WebC:webgridmessage id="messageBox" visible="False"
runat="server"></WebC:webgridmessage>
    <asp:Panel ID="panelArticleDetails" runat="server" Visible="True">
        <!--
            The panel contains all controls which are necessary to insert/edit a
new/existing
            item; it will demonstrate some special controls, such as the Infragistics
DateTime
            picker as well as the HTML editor, for composing the article body. This
panel will
            be hidden whenever the user clicked the image button. To select an image
for the
            message this panel will be hidden and the image selection panel (see below)
will be
            shown.
        -->
    <table>
    (ommitted)
```

- In the second row of the main table, right below the culture selection row, insert a new row which will allow us to define a text based unique article reference – this reference is later used in the list of pending approvals to identify an article.

```
    (ommitted)
    <tr>
        <!-- Culture Selection Box -->
        <td class="width_s alignRight">Culture</td>
        <td class="width_l">
            <asp:DropDownList ID="cultureSelection"
                runat="server" CssClass="data_input width_xl" AutoPostBack="True">
            </asp:DropDownList></td>
    </tr>
    <tr>
        <!-- Unique Article Key -->
        <td class="width_s alignRight">Article Reference</td>
        <td class="width_l">
         <asp:textbox id="txtArticleRef" onblur="this.className='data_input width_xl';"
             onfocus="this.className='data_input_over width_l';"
                runat="server" CssClass="data_input width_xl" MaxLength="50">
             </asp:textbox>
        </td>
    </tr>
    <tr>
    (ommitted)
```

- At the bottom, right before the main table is being closed, insert a new row which will contain the button to select the associated categories for the article.

```
(ommitted)
<tr>
    <!-- Articles Categories -->
    <td class="width_s alignRight">Categories</td>
    <td class="width_xl">
        <WebC:webcategorypopupbutton id="categorySelection" runat="server">
        </WebC:webcategorypopupbutton>
    </td>
</tr>
</table>
<!-- initialize the HTML editor (control id txtArticle) -->
<script language="javascript" defer>
    editor_generate('<%=ID2Name(Me.ClientID)%>:txtArticle');
</script>
(ommitted)
```

- We also need a new button between the Save –and Back button – this button will allow us to publish/ unpublish the current KnowledgeBase Article. In the table that specifies the command buttons, add a new column containing the Publish button.

```
(ommitted)
<table class="filter_btns" cellSpacing="0" cellPadding="0" border="0">
<tr>
    <td>
        <asp:button id="btnConfirm"
onmouseover="this.className='btn_m_color_over';"
onmouseout="this.className='btn_m_color';"
runat="server" CssClass="btn_m_color" Text="Confirm">
</asp:button></td>
 <td>
     <asp:button id="btnPublish"
onmouseover="this.className='btn_m_color_over';"
onmouseout="this.className='btn_m_color';"
runat="server" CssClass="btn_m_color" Text="Publish">
</asp:button></td>
    <td>
        <asp:button id="btnBack"
onmouseover="this.className='btn_m_color_over';"
onmouseout="this.className='btn_m_color';"
runat="server" CssClass="btn_m_color" Text="Back">
</asp:button></td>
</tr>
</table>
(ommitted)
```

- Now we just need to declare the category selection panel at the bottom of the file. The category selection panel will be shown whenever the user clicks on the category selection button.

```
    (ommitted)
    <asp:panel id="panelImageSelection" runat="server" Visible="False">
        <!-- image selection control panel; will only be shown when the user clicked on
             the image selection button in the details panel. -->
     <WebC:webimageselection id="controlImageSelection" runat="server">
        </WebC:webimageselection>
    </asp:panel>
    <asp:panel id="panelCategorySelection" runat="server" Visible="False">
        <!-- category selection control; will only be shown when the user clicked on
             the category selection button -->
     <WebC:webcategoryselection id="controlCategorySelection" runat="server">
        </WebC:webcategoryselection>
    </asp:panel>
```

- In order to use the approval engine that the PRIMAVERA WebCentral Platform provides we'll need references to the assemblies

  - **Primavera.Platform.Services.Entities**
  - **Primavera.Platform.Services.Business**

  to the project **Primavera.KnowledgeBase.WebUI**. These assemblies can be found in the **bin** folder of the PRIMAVERA WebCentral SDK installation path.

- To shorten up some variable type declaration, let's import these assemblies at the top of the code-behind file **ArticleAdminDetail.ascx.vb**.

```
    Imports Primavera.Platform.WebUI.Controls
    Imports Primavera.Platform.Services.Entities
    Imports Primavera.Platform.Services.Business
```

- Like already done before, we declare the controls borrowed by the PRIMAVERA WebCentral Platform in class **ArticleAdminDetail**. Note that these variables might already be declared by the Visual Studio designer and that you might remove these declarations from the file **ArticleAdminDetail.ascx.designer.vb**.

```
   (omitted)
   'declare user controls referenced from the PRIMAVERA WebCentral Platform
    'here in the code behind file, using the exact variable name as stated in the ascx
    Protected WithEvents WebImageSelectionPopUpButton1 As
     Primavera.Platform.WebUI.WebImageSelectionPopUpButton
    Protected WithEvents controlImageSelection As
Primavera.Platform.WebUI.WebImageSelection
    Protected WithEvents categorySelection As
Primavera.Platform.WebUI.WebCategoryPopUpButton
    Protected WithEvents controlCategorySelection As
Primavera.Platform.WebUI.WebCategorySelection
    Protected WithEvents messageBox As Primavera.Platform.WebUI.Controls.WebGridMessage
```

```
(omitted)
```

- We also will repeatedly need the unique module –and approval type identifier, so it's good practise to declare a variable with these values. Please make sure that the below values match with the ones in your project. See the **Primavera.KnowledfeBase.Module** project for reference.

```
'private constants
Private moduleId As New Guid("39af2e6e-e68e-4111-9410-24de9682421d")
Private moduleApprovalTypeId As New Guid("448E2B4B-979E-4c1a-878E-769BD0D7FEFD")
```

- In several places we will show messages to the user, using the MessageBox component provided by the PRIMAVERA WebCentral Platform. To easen up the usage, we'll provide a method which does the property initialization and visualization stuff.

```
'<summary>
'  The following method will show the message box control and initialize
'  the controls properties, like title, message and style.
'</summary>
Public Sub ShowMessage( _
    ByVal Type As Platform.WebUI.Controls.WebGridMessageType, _
    ByVal Title As String, _
    ByVal Message As String)

    messageBox.Visible = True
    messageBox.MessageType = Type
    messageBox.MessageTitle = Title
    messageBox.MessageText = Message
    messageBox.Refresh()
End Sub
```

- A KnowledgeBase Article must be associated at least to one category – this is a requirement which we'll need to validate before we update the record in the database. Implement a new method which checks this precondition and raise an exception if the article does not meet the validation rules.

```
'<summary>
'  The following method will validate the current artucle before it
'  is going to be written to database.
'</summary>
Private Sub ValidateBusinessEntity()

    Dim validationMessage As String = String.Empty
    If BusinessEntity.Categories.Count <= 0 Then _
        Throw New System.Exception("The Knowledge Base Article " _
            & "needs to be associated to at least one category")

End Sub 'ValidateArticle
```

- We already implemented the **ShowBusinessEntity** method a while ago to initialize the user interface controls with the values from the business entity object. However, we now have a new attribute, the article reference, which we'll need to initialize. Insert a new line to this method, which will initialize the article reference textbox.

```
    '<summary>
    '  The following method will be called whenever it is necessary to initialize
    '  the user control with values from the currently loaded business entity. This
    '  is basically everytime the case, when a user edits an existing record.
    '</summary>
    Private Sub ShowBusinessEntity()

        txtArticleRef.Text = Me.BusinessEntity.Article
        txtAutor.Text = Me.BusinessEntity.Author
        txtArticle.Text = Me.BusinessEntityCulture.Body
        txtDate.Value = Me.BusinessEntity.Date
        txtTitle.Text = Me.BusinessEntityCulture.Title
        txtImage.Text = Me.BusinessEntity.Image

        'update image selection control
        WebImageSelectionPopUpButton1.ImageHtml = Me.BusinessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'ShowBusinessEntity
```

- The method **UpdateBusinessEntity** implements the opposite way – writing the values from the user interface to the business entity object. Here, we also need to serialize the article reference. This is also a good place to do the business entity validation, as this method is usually called whenever the entity object is being serialized to database.

```
    '<summary>
    '  The following method will be called whenever it is necessary to write all
changes
    '  down to database. This method will update the existing record, which was edited,
or
    '  inserts a new record to the database.
    '</summary>
    Private Sub UpdateBusinessEntity()

        Me.BusinessEntity.Article = txtArticleRef.Text
        Me.BusinessEntity.Author = txtAutor.Text
        Me.BusinessEntity.Date = txtDate.Value
        Me.BusinessEntity.Image = txtImage.Text
        Me.BusinessEntityCulture.Title = txtTitle.Text
        Me.BusinessEntityCulture.Body = txtArticle.Text
        Me.ValidateBusinessEntity()

        Dim businessLayer As New Business.Articles
        businessLayer.Update(WebContext.User.EngineContext, Me.BusinessEntity)
    End Sub 'UpdateBusinessEntity
```

The method **CreateBusinessEntity** is called whenever the user chose to create a new KnowledgeBase Article. This method is responsible for initializing the internally used business entity object. Because of the new attribute, as well as the categories, we'll need to update this code to initialize these properties.

```
'<summary>
'  The following method will be called whenever the user wishes to create
'  a new record. This method initializes a new business entity object.
'</summary>
Private Sub CreateBusinessEntity()

    'initialize new business entity
    Dim businessEntity As New Entities.Article
    businessEntity.ID = Guid.NewGuid()
    businessEntity.Article = String.Empty
    businessEntity.Author = WebContext.User.UserName
    businessEntity.Date = DateTime.Now
    businessEntity.Image = String.Empty

    Dim businessEntityCulture As New Entities.ArticleCulture
    businessEntityCulture.ArticleID = businessEntity.ID
    businessEntityCulture.CultureID = m_activeCulture
    businessEntity.ArticleCultures.Add(businessEntityCulture)

    'categories
    categorySelection.SelectedCategories = New ArrayList
    categorySelection.SetConfigurationParameters()

    'initialize image button
    WebImageSelectionPopUpButton1.ImageHtml = businessEntity.Image
    WebImageSelectionPopUpButton1.SetConfigurationParameters()

    Me.BusinessEntity = businessEntity
    Me.BusinessEntityCulture = businessEntityCulture

End Sub 'CreateBusinessEntity
```

- Same procedure with the method **EditBusinessEntity**. We'll need to initialize the category selection control with the categories associated to the loaded KnowledgeBase Article as well initialize the Publish buttons caption in dependency of the current publishing state (the caption of this button toggles between **Publish** and **Unpublish**)

```
'<summary>
'  The following method will be called to initialize the internally used business
'  entity object for being edited. It will load the specified record from database;
'  the record's primary key needs to be set before.
'</summary>
Private Sub EditBusinessEntity()
```

```
        Dim businessLayer As New Business.Articles
        Me.BusinessEntity = businessLayer.Edit(WebContext.User.EngineContext,
Me.ArticleId)
        Me.BusinessEntityCulture =
Me.BusinessEntity.ArticleCultures.GetItemByCultureID(m_activeCulture)

        If Me.BusinessEntityCulture Is Nothing Then
            Me.BusinessEntityCulture = New Entities.ArticleCulture
            Me.BusinessEntityCulture.ArticleID = BusinessEntity.ID
            Me.BusinessEntityCulture.CultureID = m_activeCulture
            Me.BusinessEntity.ArticleCultures.Add(BusinessEntityCulture)
        End If

        'categories
        categorySelection.SelectedCategories = New ArrayList
        For Each Category As EntityCategory In Me.BusinessEntity.Categories
            categorySelection.SelectedCategories.Add(Category.CategoryID)
        Next
        categorySelection.SetConfigurationParameters()

        'initialize image button
        WebImageSelectionPopUpButton1.ImageHtml = Me.BusinessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

        'show the correct text for the publishing/unpublishing button
        btnPublish.Text = IIf(Me.BusinessEntity.Published, "Unpublish", "Publish")

        'show information whenever the current article is waiting for approval
        If Me.Mode = VisualizationMode.Edition _
            AndAlso Not Me.BusinessEntity Is Nothing _
            AndAlso Me.BusinessEntity.Locked Then

            Me.ShowMessage(WebGridMessageType.Info, String.Empty, _
                "An approval is pending for this article")

        End If

    End Sub 'EditBusinessEntity
```

- The next method **GetApprovalID** will try to query the approval typ id that needs to be applicated to the current article. The approval type depends on the categories that were associated to the article and which approval rules have been associated to these categories. This method will return an empty Guid if no approval rule has been associated to the categories, otherwise the approval rule id.

```
    '<summary>
    '  The following method will return the approval identifier for the specified
    '  categories.
    '</summary>
    Private Function GetApprovalRuleID(ByVal objCategories As _
```

```
Primavera.Platform.Services.Entities.EntityCategories) As Guid
        Dim objFields As New Primavera.Platform.Engine.Entities.Fields
        Dim objCat As Primavera.Platform.Services.Entities.EntityCategory
        Dim objBusiness As New Business.Articles

        For Each objCat In objCategories
            objFields.Add(objCat.CategoryName, objCat.CategoryID)
        Next

        Return objBusiness.GetApprovalRuleID(WebContext.User.EngineContext, objFields)
    End Function 'GetApprovalRuleID
```

- The next method will check if the currently edited KnowledgeBase Article is locked, which means, it was published and sent to approval, but has not been approved yet. In this case, the user receives a message informing him on this matter.

```
    '<summary>
    '  The following method will check if the specified KnowledgeBase article
    '  is currently locked.
    '</summary>
    Private Function ArticleIsLocked(ByVal guidID As Guid) As Boolean
        Dim objBusiness As New Business.Articles
        Return DirectCast(objBusiness.EditField( _
WebContext.User.EngineContext, guidID, "Locked"), Boolean)
    End Function 'ArticleIsLocked
```

- The following method will be called whenever an article has been sent to approval. This flag in the business entity object, in conjunction with the Published flag, allows us to easily determine the state of the article.

```
    '<summary>
    '  The following method will set the locked flag and update the
    '  approval identifier for the specified article
    '</summaery>
    Private Sub LockArticle(ByVal guidArticleID As Guid, ByVal guidApprovalID As Guid)

        Dim objBusiness As New Business.Articles
        objBusiness.Lock(WebContext.User.EngineContext, guidArticleID, guidApprovalID)
        ShowMessage(WebGridMessageType.Info, String.Empty, _
            "The new article has been submitted to approvation.")

    End Sub 'LockArticle
```

- Now it’s time to implement the method which actually starts the approval for the business entity object. This method will sent the article for approval to the responsible user and broadcasts notification messages, if so configured.

```
    '<summary>
    '  Start Article Approval
    '</summary>
```

```
    Private Function StartApproval(ByVal guidArticleID As Guid, _
ByVal guidApprovalRuleID As Guid, ByRef guidApprovalID As Guid) As ApprovalResponse
        Dim objBusiness As New Platform.Services.Business.ApprovalServices
        Dim objRequest As New ApprovalRequest
        With objRequest
            .ApprovalID = guidApprovalID
            .ApprovalRuleID = guidApprovalRuleID
            .ModuleID = moduleId
            .TypeID = moduleApprovalTypeId
        End With

        Return objBusiness.StartApproval(WebContext.User.EngineContext, objRequest)
    End Function 'StartApproval
```

- The next method is a high-level method, which is called directly from the Publish button event handler. This method actually checks if the article is subject to approval and respectively starts the approval process calling the **StartApproval** method. The article will be directly saved and publish if it does not have any category with approval rule associated.

```
    Private Function ApprovalAndPublish(Optional ByRef blnLocked As Boolean = False) As
Boolean

        'instanciate business object
        Dim objBusiness As New Business.Articles

        'Default result (everything OK)
        ApprovalAndPublish = True

        'Get applicable approval rule
        Dim guidApprovalRuleID As Guid = GetApprovalRuleID(BusinessEntity.Categories)

        'Subject to approval?
        If guidApprovalRuleID.Equals(Guid.Empty) Then
            'Publish the KnowledgeBase Article
            objBusiness.Publish(WebContext.User.EngineContext, BusinessEntity.ID)
        Else
            'Register the approval
            Dim guidApprovalID As Guid = Guid.NewGuid
            Dim objResponse As ApprovalResponse = StartApproval(BusinessEntity.ID, _
                guidApprovalRuleID, guidApprovalID)
           'The approval succeeded?
            If objResponse.Executed Then
                'If the approval whas not auto ended, lock the content
                If (Not objResponse.Finished) Then '
                    LockArticle(BusinessEntity.ID, guidApprovalID)
                    blnLocked = True
                Else
                    objBusiness.Publish(WebContext.User.EngineContext,
BusinessEntity.ID)
                End If
```

```
            Else
                ApprovalAndPublish = False 'Approval failed
            End If
        End If
    End Function 'ApprovalAndPublish
```

- Even though we hid the message box in the ASCX file we need to make sure that it's hidden with every server roundtrip to avoid that expired information messages are shown to the user. Place the following line in the **Page_Load** event handle.

```
        'hide the message box
        messageBox.Visible = False
```

- Create a new event handler for the category selection control. This method will be called when the user associated categories to the article and clicked on the Confirm button. We need to update the business entity object's category properties with the associated categories.

```
    Private Sub categorySelection_CategorySelected( _
        ByVal selectedCategories As System.Collections.ArrayList) _
        Handles categorySelection.CategorySelected

        Dim entityCategories As New
Primavera.Platform.Services.Entities.EntityCategories
        Dim entityCategory As Primavera.Platform.Services.Entities.EntityCategory

        For Each category As Guid In selectedCategories
            entityCategory = New Primavera.Platform.Services.Entities.EntityCategory
            entityCategory.CategoryID = category
            entityCategories.Add(entityCategory)
        Next
        Me.BusinessEntity.Categories = entityCategories

    End Sub 'WebCategoryPopUpButton1_CategorySelected
```

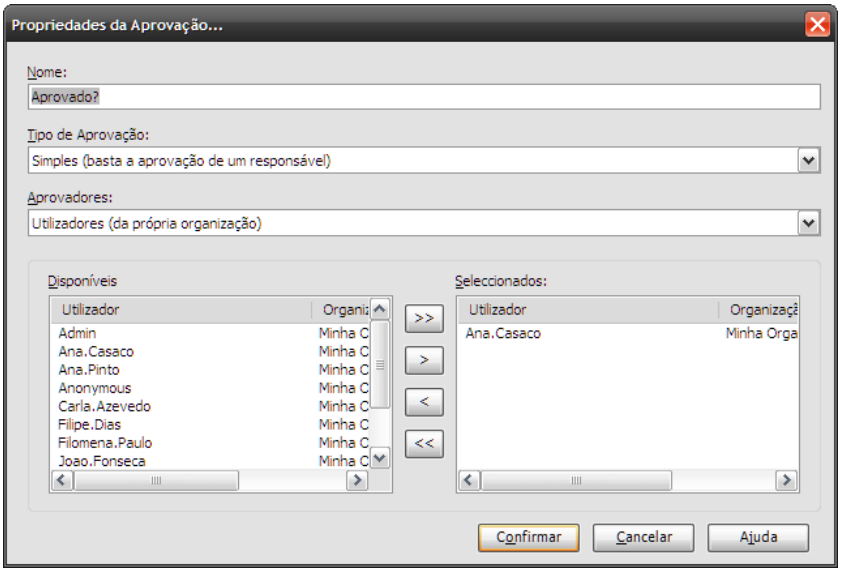
- The last step is implementing the code for the Publish button, which actually treats the publishing and unpublishing process, and makes use of the helper methods, which we coded before.

```
    Private Sub btnPublish_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnPublish.Click

        Try
            Dim businessObject As New Business.Articles
            If Me.BusinessEntity.Published Then
                businessObject.UnPublish(WebContext.User.EngineContext,
BusinessEntity.ID)

                'refresh the business entity object from database
                Me.ArticleId = BusinessEntity.ID
```

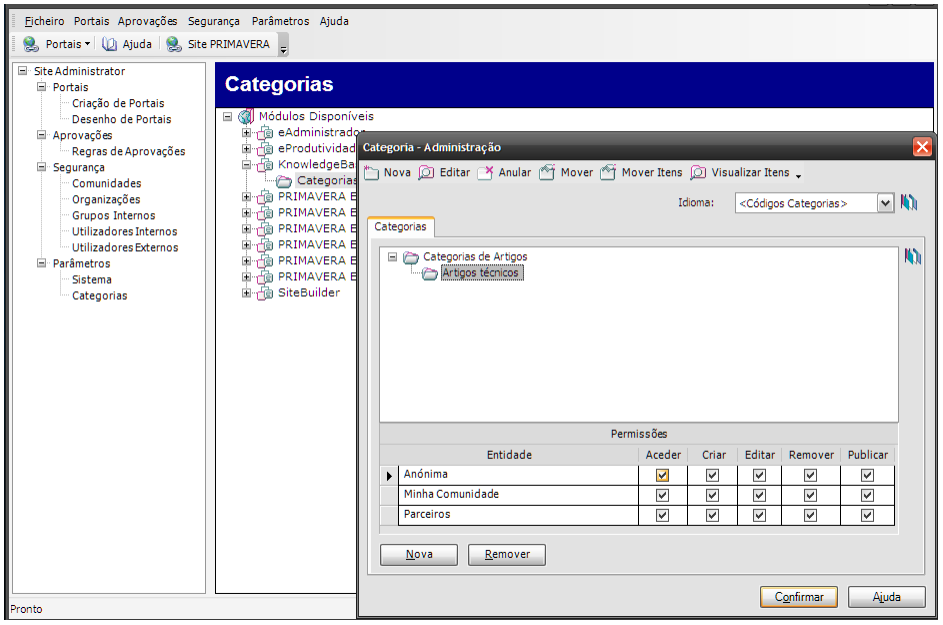
```
                Me.EditBusinessEntity()

                'show publication message
                Me.ShowMessage(WebGridMessageType.Info, String.Empty, _
                    "Publication has been cancelled with success")
            Else
                'update the business entity from the current user interface's values
                Me.UpdateBusinessEntity()

                Dim locked As Boolean = False
                Dim returnMessage As String = String.Empty
                Dim result As Boolean = Me.ApprovalAndPublish(locked)
                If locked Then
                    Me.ShowMessage(WebGridMessageType.Info, String.Empty, _
                        "Article has been saved and waits for approval.")
                Else
                    Me.ShowMessage(WebGridMessageType.Info, String.Empty, _
                        "Publication has been executed with success.")
                End If

                'refresh the business entity object from database
                Me.ArticleId = BusinessEntity.ID
                Me.EditBusinessEntity()
            End If

        Catch ex As Exception
            Me.ShowMessage(WebGridMessageType.Error, String.Empty, ex.Message)

        End Try
    End Sub
```

Step 7.7 – Testing the Approval and Category Support

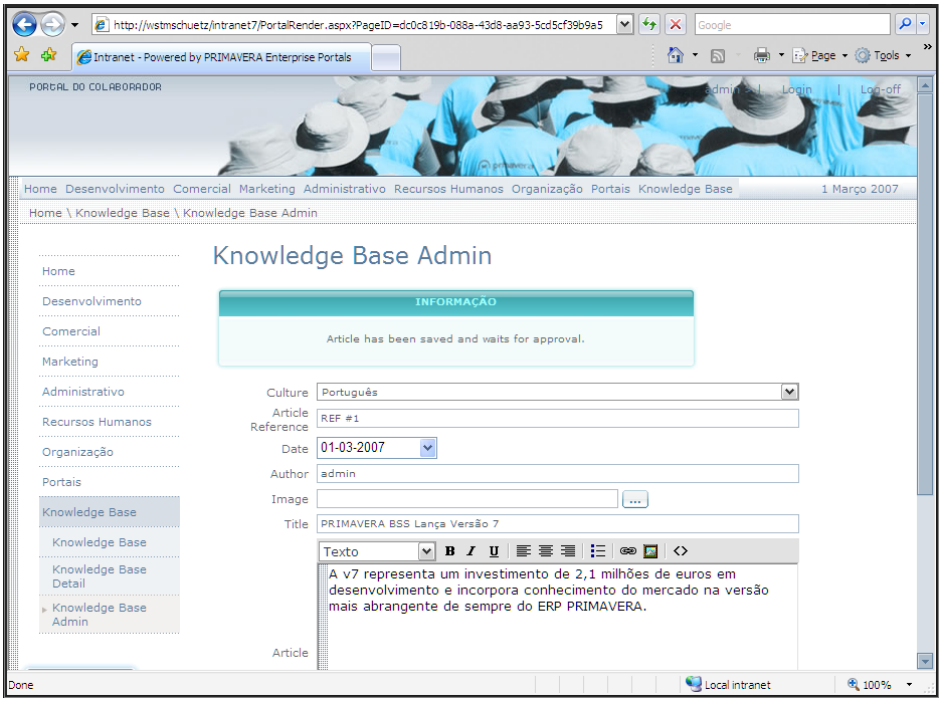
- Use the Visual Studio 2008 IDE to compile the solution in DEBUG mode.
- Close the Visual Studio 2008 IDE and run the **eModuleDeploy** utility, which you will find in the **Tools** folder of the PRIMAVERA WebCentral SDK.
- Indicate the complete filename of our module's solution and start the deployment process.
- In appendix 19 [page 157] you will find the SQL script that is necessary to reflect the changes in the database. Run this script over the WebCentral database using SQL Query Analyzer.
- After the deployment process has completed, launch the Site Administrator application and specify a new approval rule for the KnowledgeBase Articles. Specify **ana.casaco** as responsible for approving the articles.



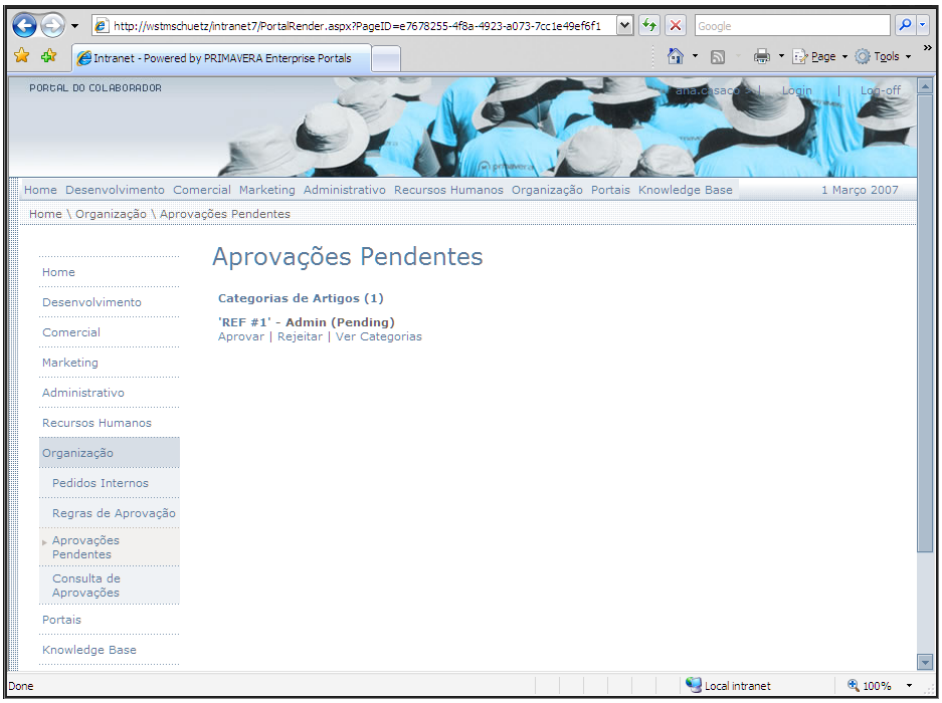
- Confirm the new Approval Rule and associate it to the KnowledgeBase Article category Artigos Técnicos. From now on, all articles that were associated to this category will be subject to being approved by ana.casaco.



- Launch your web browser and navigate to the KnowledgeBase administration component. Create a new KnowledgeBase article and associate it to the category that is subject to approval. Try publishing the article and you'll see that the article has been sent to approval.


http://wstmschuetz/intranet7/PortalRender.aspx?PageID=dc0c819b-088a-43d8-aa93-5cd5cf39b9a5
Intranet - Powered by PRIMAVERA Enterprise Portals
PORTAL DO COLABORADOR
Login
Log-off
Home Desenvolvimento Comercial Marketing Administrativo Recursos Humanos Organização Portais Knowledge Base
1 Março 2007
Home \ Knowledge Base \ Knowledge Base Admin
Home
Desenvolvimento
Comercial
Marketing
Administrativo
Recursos Humanos
Organização
Portais
Knowledge Base
Knowledge Base
Knowledge Base Detail
Knowledge Base Admin
Knowledge Base Admin
INFORMAÇÃO
Article has been saved and waits for approval.
Culture
Português
Article Reference
REF #1
Date
01-03-2007
Author
admin
Image
Title
PRIMAVERA BSS Lança Versão 7
Texto
A v7 representa um investimento de 2,1 milhões de euros em desenvolvimento e incorpora conhecimento do mercado na versão mais abrangente de sempre do ERP PRIMAVERA.
Article
Done
Local intranet
100%

- Navigate to the list of published articles, just to check that the new article is not visible yet – it should not appear as Ana Casaco still needs to approve it.
- Now, log in as ana.casaco and navigate the items that need her approval. You should see at least the article that we created before. Click on the **Approve** link to finally publish the article.



- If you now access the list of available KnowledgeBase Articles you should see the article that we have created before.

Step 8 – Crystal Reports Integration

You'll probably encounter situation where simple HTML output to the browser won't be enough to present dynamic, database driven content to the user. You might come to the point where you need to create dynamic PDF files on the server and send the generated file to the client. The salary receipt generation of the Human Resources Module of the WebCentral is a good example of where this is already implemented.

In this chapter we'll see how easy it is to generate dynamic PDF files using the WebCentral SDK. We'll add a new component to our already existing solution, which generates a PDF report with all existing KnowledgeBase Articles. You can use the here presented code as base for you own solution.

Step 8.1 – Creating a new component for reporting

Like already mentioned, we're going to demonstrate the dynamic PDF creation by implementing a new component for the KnowledgeBase module. You already know the step's to do so by completing the previous chapter's, so here's a good chance to check your knowledge.

- Open the project **Primavera.KnowledgeBase.Modules** and edit the file **ComponentsIDs.vb**

- Add a new shared read-only property named KnowledgeBaseReport, which returns a newly defined Guid.

```
Public Shared ReadOnly Property KnowledgeBaseReport() As Guid
    Get
        Return New Guid("{4a89d1e4-17ec-416c-87d3-d3e97529229f}")
    End Get
End Property
```

- Now that we've got the new components identifier, let's publish the new component to the WebCentral Platform and specify the component's properties. Open the file **Components.vb** and locate the method **List**. Add the following line the the already existing implementation.

```
        ElseIf ComponentType = ComponentType.Publishing Then
            objCol.Add(ComponentsIDs.KnowledgeBaseArticle, "Article Detail")
            objCol.Add(ComponentsIDs.KnowledgeBaseArticleList, "Article List")
            objCol.Add(ComponentsIDs.KnowledgeBaseReport, "Article Report")
```

- Still in the file **Components.vb**, locate the method **Edit** and insert the section which will return essential information about the new component.

```
        ElseIf ComponentID.Equals(ComponentsIDs.KnowledgeBaseReport) Then
            objComp.Name = "Article Report"
            objComp.Description = "Component for creating article reports"
            objComp.ComponentClass = "ArticleReport"
            objComp.RequiresEnterprise = False
            objComp.IconIndex = 0
            objComp.BackgroundOpaque = True
            objComp.AllowsFrame = True
            objComp.ComponentSecurity = False
```

- We already have done everything to inform the WebCentral Platform about the new component we're going to provide. All that is left is creating the component and the Crystal Report file itself. To do that, open the project **Primavera.KnowledgeBase.WebUI** and insert a new Web Usercontrol to the folder **Components**. You should already be able to guess the components name from the previous step. Call it **ArticleReport.ascx**.
- Open the new components HTML view and insert the following two lines. These two lines will define a message box component, which will be used as error message, just in case something went wrong during the PDF report creation.

```
<%@ Register TagPrefix="WebC" TagName="WebGridMessage"
Src="../Primavera.Platform/WebGridMessage.ascx" %>
<WebC:webgridmessage id="messageBox" runat="server"
visible="False"></WebC:webgridmessage>
```

Now we come to the real interesting part – the code that actually will pick up an already existing Crystal Report file and generate a PDF which is than sent to the client's browser. You actually can use this code as snippet and copy and paste it in your own project. You only should take care about the database connection parameters, the report's filename as well as the selection formulas you need.

- Change the user controls base class to **WebComponentBase**.

```
    Partial Public Class ArticleReport
            Inherits Primavera.Platform.WebUI.WebComponentBase
```

➢ Open the new component's code behind file and insert the following code.

```
    '<summary>
    '  The following method is responsible for correctly creating the
    '  report in PDF format. The function will return the complete
    '  filename for the report on success.
    '</summary>
    Private Function PrintReport() As String

        Dim reportArticles As Primavera.Platform.Services.Entities.CrystalReport = _
New Primavera.Platform.Services.Entities.CrystalReport
        Dim reportFileName As String =
Server.MapPath("Modules/Primavera.KnowledgeBase/Report.rpt")

        reportArticles.Location = reportFileName
        reportArticles.Copies = 1
        reportArticles.Destination =
Primavera.Platform.Services.Entities.ReportDestination.Preview
        reportArticles.Format = Primavera.Platform.Services.Entities.ReportFormat.PDF
        reportArticles.Title = "Knowledge Base Articles"

        reportArticles.LogonInfo.Login = "sa"
        reportArticles.LogonInfo.Password = "sasa"
        reportArticles.LogonInfo.Server = "WSTMSCHUETZ\SQLEXPRESS"
        reportArticles.LogonInfo.Database = "EPRIMAVERA"

        reportArticles.SelectionFormula = "{KNB_ArticleCultures.CultureID}=""{" _
            + WebContext.User.Culture.CultureID.ToString() + "}"""

        Dim engineCtx As Platform.Engine.Entities.EngineContext =
Me.WebContext.User.EngineContext
        Dim serviceProxy As Primavera.Platform.Services.Business.CrystalReports = _
New Primavera.Platform.Services.Business.CrystalReports
        Dim resultFilename As String = _
serviceProxy.ExportReport(engineCtx, reportArticles, Guid.NewGuid.GetHashCode())
        If resultFilename Is Nothing Then
            messageBox.Visible = True
            messageBox.MessageType = Platform.WebUI.Controls.WebGridMessageType.Error
            messageBox.MessageTitle = "Primavera.KnowledgeBaseArticleReport"
            messageBox.MessageText = _
String.Format("An unexpected error occurred while creating the requested report ({0})",
_
                 reportArticles.Location)
            messageBox.Refresh()
            Return String.Empty
        End If
```

```
        Return Me.WebContext.Portal.PortalUrl + "/" + resultFilename

    End Function 'PrintReport
```

- Please make sure that you substitute the parameters like the SQL logon info with the correct values. It surely is good practise to place these configuration settings into secure and easy maintainable configuration files, like for example the **web.config** file**.** You can find valuable information about this in the Microsoft Developer Network.
- As you can see, this method will return the absolute URL path of the generated PDF report. All we need to do now is call this method in the Page_Load handler, and redirect the HTTP response to the file generated. Locate the **Page_Load** method and insert the following lines of code:

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Dim reportFilename As String = PrintReport()
        If Not reportFilename.Equals(String.Empty) Then
            Response.Redirect(reportFilename)
            Response.End()
        End If

    End Sub
```

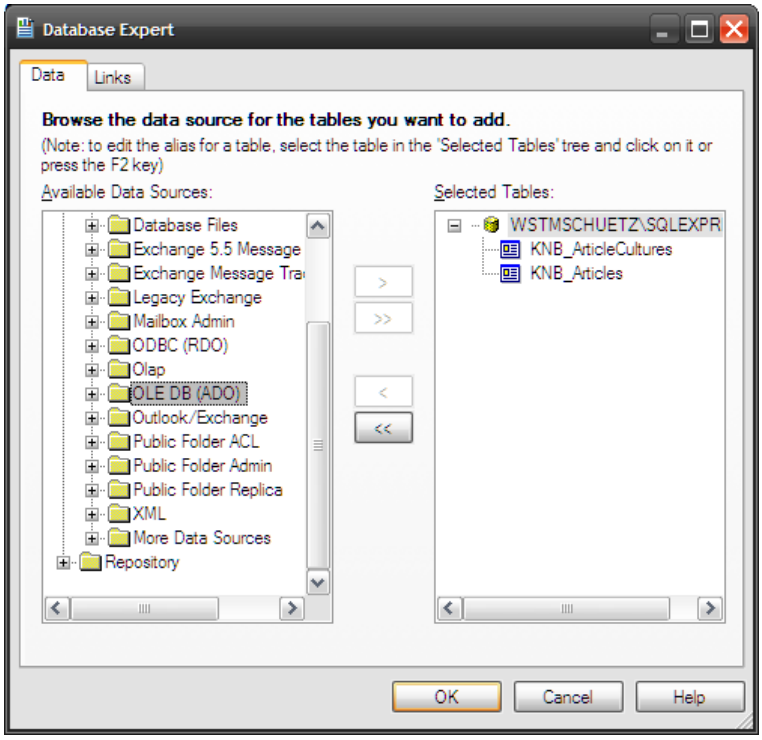
We've now added a new component to our module, which dynamically creates a PDF report having all our KnowledgeBase articles in the currently active culture, and sents this report to the client's browser. Your code now should be compilable and you should be able to deploy the module to the WebCentral Platform.

What we still need to do, is create the Crystal Report Template file and store it at the place where the code expects it to be (C:\inetpub\wwwroot\WebCentral\Modules\Knowledge Base\Report.rpt). We'll discuss this in the next step.

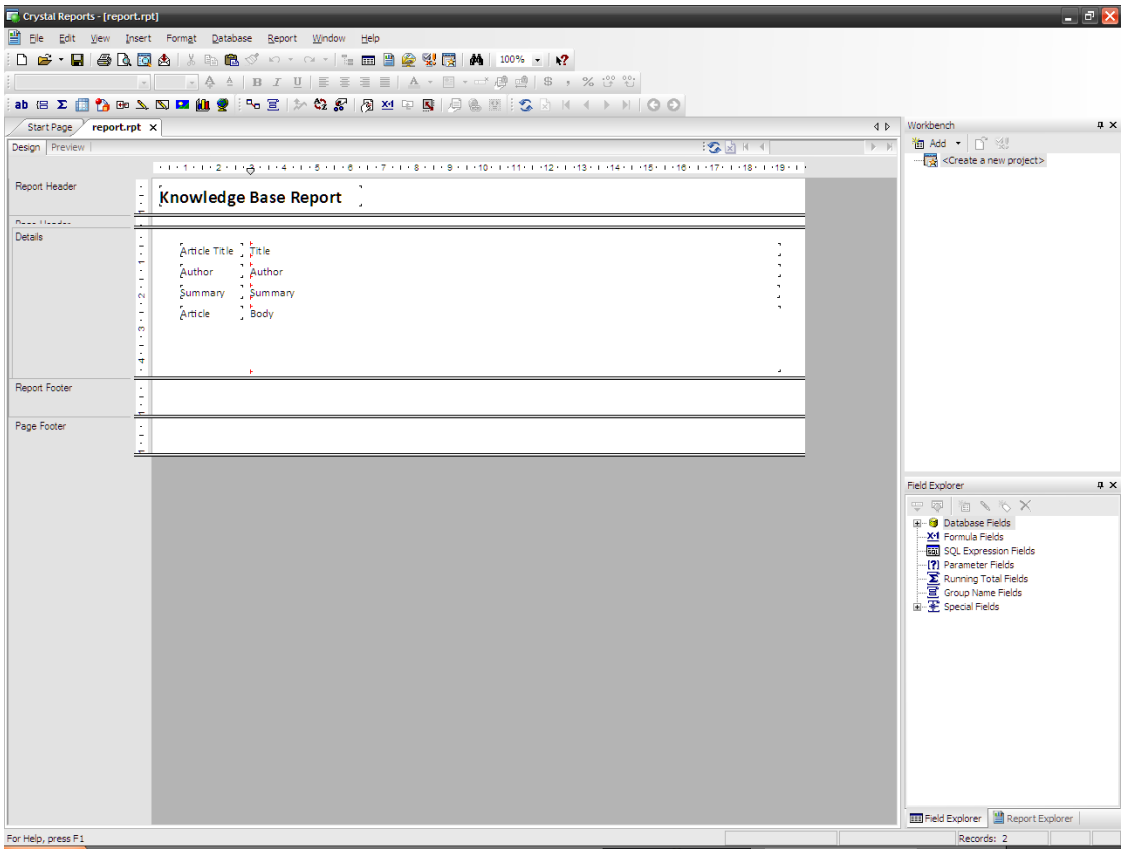
Step 8.2 – Creating the Crystal Report document

The WebCentral Platform uses componens from Crystal Reports 2008 to dynamically create PDF reports. For that reason, the Crystal Reports 2008 designer should be your tool of choice to create the report template files. However, you can use any other version, as long as the created file is compatible with the expected format.

To create the report template file, open the designer and connect to the database that is used by the PRIMAVERA WebCentral Platform. You can use the OLE DB (ADO) connection provider for SQL Server to create a new database connection. Select the tables **KNB_ArticleCultures** and **KNB_Articles** for being used in the report.



In the reports design mode, drag and drop fields Title, Author, Summary and Body to the detail section of the report. Check the preview pane for how the compiled will look like.



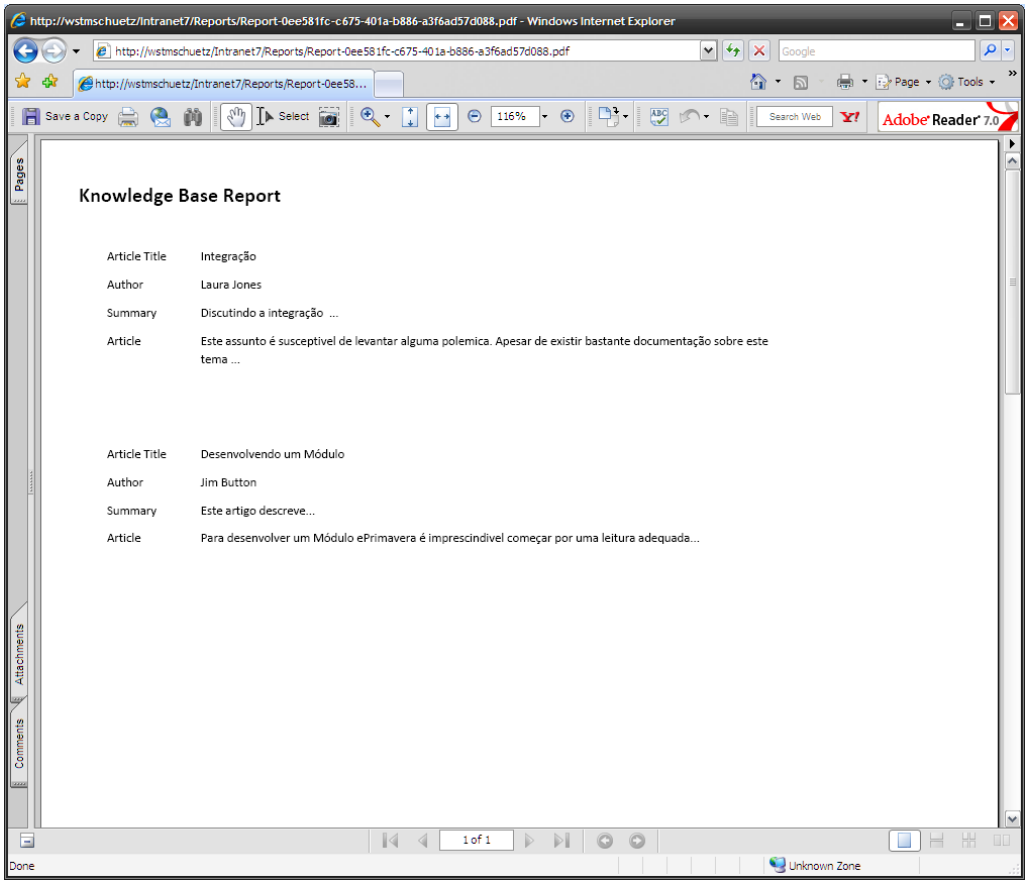
If you have completed design, store the file to the directorty where the component expects it to be. In our example, it expects it to be in the folder

C:\inetpub\wwwroot\WebCentral\Modules\Knowledge Base

and its name should be **report.rpt**.

Step 8.3 – Testing the Reporting component

- If you haven't already done it, use the Visual Studio 2008 IDE to compile the solution in DEBUG mode.
- Close the Visual Studio 2008 IDE and deploy the solution using the eModuleDeploy utility.
- Use the Site Administrator utility to create a new page **Article Report** in the portal that you previously used to test the KnowledgeBase module.
- Place the new **Article Report** in this new page.
- Check-in the changes and publish the portal.
- Navigate to the page that contains the Article Report component. The component should create the dynamic PDF document and render it to your browser.



PRIMAVERA ERP Integration

The development of models that integrate the PRIMAVERA ERP is just like developing any other type of module. However you will need to take at least two aspects in consideration.

The access to the PRIMAVERA ERP database

You never should take for granted that the PRIMAVERA ERP database lives on the same database server than the database for PRIMAVERA WebCentral. The PRIMAVERA ERP database location is defined while initializing the PRIMAVERA ERP instance using the PRIMAVERA ERP Administration tool. The PRIMAVERA WebCentral module should always use PRIMAVERA ERP platform engine to initialize the session and open the required enterprise database.

Usage of ERP Web Business Services to access PRIMAVERA ERP engine

Since v8.0 SR1 WebCentral, a new way to communicate with PRIMAVERA ERP is available. This is and will be the preferred method from this version on. It is based on web services, which are created after running the *Erp Web Business Services* setup available on the WebCentral DVD.

This new layer allows for 64-bit project configurations/compilations and greater compatibility between ERP versions and WebCentral as the Webservices are created at runtime using current interops working on the machine. This new method removes most likely conflicts between the WebCentral components and the existing ERP version.

Depeding on the estimated workload, network configuration and final client dimension it is possible to configure one machine for the ERP Web Services and another for Webcentral or use the same machine for both tasks, this allows for a flexible system configuration and a better resource management according to client needs.

It is also quite easier to develop components which use ERP data, than in previous WebCentral/Enterprise Portals versions.

Please note that this is available from ERP version 7.55 SR2 or newer.

Overview

The PRIMAVERA WebCentral Platform provides one module which will support you while developing customized modules that integrate the PRIMAVERA ERP.

First off, if there are any existing references to Interops in your solution they **must** be removed. Next, add four new references to your project, they are: "Primavera.ERPOnline.Bso.Proxy.dll" and "Primavera.ERPOnline.Services.Proxy.dll".These dll's can be found after installing the *Erp Web Business Services* under its "\ProxyClient" folder (ex: "C:\inetpub\wwwroot\ERPWebBusinessServices\ProxyClient"). The remaing two references are "Primavera.ERP.Entities" and "Primavera.ERP.Business", both can be found in the WebCentral SDK Folder.

Creating a new Web component for ERP Data Access

Again we can use what we have already learned and create a new component for showing ERP Data. This time we are going to use the base solution provided by Application Builder.

- Create a new Solution in the Application builder
- Edit the created solution and publish the "programming logic"

- Open the generated solution in Visual Studio.

We are now going to create a new component named **ERPSDK**. After opening the solution follow these steps.

- Open the project **Entities** in the solution and edit the file **StaticIDsBase.vb**
- Add a new value to the **ComponentsEnum** Enum.

```
    Public Enum ComponentsEnum
        BizEntityViewer
        BizModelExplorer
        TreeModelExplorer
        ERPSDK
    End Enum
```

- In the same file find property named components and add a new item with a new guid.

```
Public Shared ReadOnly Property Components(ByVal Item As ComponentsEnum) As Guid
        Get
            Select Case Item
                Case ComponentsEnum.BizEntityViewer
                    Return New Guid("{A7A37221-447E-4654-89B0-E79F2E31F182}")
                Case ComponentsEnum.BizModelExplorer
                    Return New Guid("{A7A37221-447E-4654-89B0-E79F2E31F183}")
                Case ComponentsEnum.TreeModelExplorer
                    Return New Guid("{A7A37221-447E-4654-89B0-E79F2E31F184}")
                Case ComponentsEnum.ERPSDK
                    Return New Guid("{A7A37221-447E-4654-89B0-E79F2E31F185}")
            End Select
        End Get
    End Property
```

- Now that we've got the new components identifier, let's publish the new component to the WebCentral Platform and specify the component's properties. Open the file **Components.vb** under the Modules project and locate the method **List**. Add the following line the the already existing implementation.

```
 ElseIf ComponentType = ComponentType.Publishing Then
     Components.Add(Entities.Proxy.StaticIDs.Components(Entities.StaticIDs.ComponentsEn
um.BizEntityViewer), ResMan.GetString("RES_Components_BizEntityViewer"))
     Components.Add(Entities.Proxy.StaticIDs.Components(Entities.StaticIDs.ComponentsEn
um.BizModelExplorer), ResMan.GetString("RES_Components_BizModelExplorer"))
     Components.Add(Entities.Proxy.StaticIDs.Components(Entities.StaticIDs.ComponentsEn
um.TreeModelExplorer), ResMan.GetString("RES_Components_TreeModelExplorer"))
     Components.Add(Entities.Proxy.StaticIDs.Components(Entities.StaticIDs.ComponentsEn
um.ERPSDK), "ERPSDK Component")
```

- Still in the file **Components.vb**, locate the method **Edit** and insert the section which will return essential information about the new component.

```
        ElseIf ComponentID.Equals(ComponentsIDs.KnowledgeBaseReport) Then
            objComp.Name = "ERPSDK Comp"
            objComp.Description = "ERPSDK Component"
```

```
            objComp.ComponentClass = "ERPSDKComponent"
            objComp.RequiresEnterprise = True 'This options trigger the show of the
combobox which allows for Enterprise Selection
            objComp.IconIndex = 0
            objComp.BackgroundOpaque = True
            objComp.AllowsFrame = True
            objComp.ComponentSecurity = False
```

- Now all that is left is creating the component file itself. To do that, open the **WebUI** project and insert a new Web Usercontrol to the folder **Components**. Call it **ERPSDKComponent.ascx**. The component class must match the ComponentClass property inserted in the previous step. It is case sensitive.
- Open the new components HTML view and insert the following two lines. These two lines will define a message box component, which will be used as error message, just in case something went wrong during the PDF report creation.

```
<%@ Register TagPrefix="WebC" TagName="WebGridMessage"
Src="../Primavera.Platform/WebGridMessage.ascx" %>
<WebC:webgridmessage id="messageBox" runat="server"
visible="False"></WebC:webgridmessage>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<br />
<asp:GridView ID="GridView1" runat="server"></asp:GridView>
```

Now let's place some data in the inserted controls.

- Change the user controls base class to **WebComponentBase**.

```
    Partial Public Class ArticleReport
            Inherits Primavera.Platform.WebUI.WebComponentBase
```

- The following code segment shows how to invoke the PRIMAVERA ERP engine and bind the list of available items for the selected enterprise to a data grid control as well as to show the current user name for employee with code "F001".

```
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Dim erpContext As Primavera.ERP.Entities.ERPContext = Nothing
        Dim func As Primavera.ERPOnline.Bso.RhpBEFuncionario = Nothing
        Dim contactos As Primavera.ERPOnline.Bso.GcpBEContactos = Nothing

        Try
          erpContext =
Primavera.ERP.Business.Proxy.OpenEnterprise(WebContext.Instance.Parameters.ERPPlatformV
ersion, WebContext.User.EngineContext.EnterpriseCode, WebContext.User.ERPAccountUser,
WebContext.User.ERPAccountPassword, WebContext.Instance.Parameters.ERPInstance)
          func = erpContext.BSO.RecursosHumanos.Funcionarios.Edita("F001")
Label1.Text = func.Nome
          artigos = erpContext.BSO.Comercial.Artigos.LstArtigos
           GridView1.DataSource = artigos.DataSource.Tables
           GridView1.AutoGenerateColumns = True
```

```
            GridView1.DataBind()
       Catch ex As Exception
          messageBox.Visible = True
          messageBox.MessageType =
Primavera.Platform.WebUI.Controls.WebGridMessageType.Error
          messageBox.MessageTitle = "Error"
          messageBox.MessageText = "An error occurred: " + ex.Message + vbNewLine +
ex.StackTrace
          messageBox.Refresh()
       Finally
          func = Nothing
          contactos = Nothing
          Primavera.ERP.Business.Proxy.CloseEnterprise(erpContext)
        End Try
    End Sub
```

Please note the following: a good practice it is importante to guarantee when the closeEnterprise Method is called. Not doing so may lead to deadlocks and/or memory leaks.

Windows Components

Since all visualization components should always be written in web based components, probably you will never need to invoke the PRIMAVERA ERP engine in Windows Forms based components. From version 7 on, Windows Forms based dialogs are only used to configure components in the Site Administration utility. However you might need to access the PRIMAVERA ERP engine from a Windows Forms based component. Here's how it's done, presuming that you already implemented the remoting layer.

1. Create a Proxy from the Remoting assembly, using the form's member **`CreateRemoteProxy`**.
2. Obtain an **`ERPContext`**, if not already obtained, using the proxy's **`OpenEnterprise`** method.
3. Call the appropriate methods you need.
4. Release the **`ERPContext`** calling the proxy's **`CloseEnterprise`** method.

Frequently Asked Questions

What does the deployment process do?

The PRIMAVERA WebCentral SDK provides a utility that hides the process of deploying a customized module from you. However, you might need to deploy your module by hand, that is, for example, if you are not developing at the production server – there will come the time you'll need to deploy your module to the *live* server, which normally is not possible using the eModuleDeploy application.

So what does this small tool does? What should you do to deploy your module by hand?

1) **Prepare your module files to be deployed**
   Compile your module in RELEASE mode. Create a new folder on your local or pen drive and copy all files that need to be deployed to this folder. At least you should copy all assemblies, web pages and web user controls (*.dll, *.ascx, *.aspx) to this folder. Depending on your code, you also might need other files.

2) **Stop the Internet Information Services**
   To avoid that files get locked due to ongoing processes, avoid that requests to the web server are made – stop the Website in the Internet Information Services Administration Console.

3) **Copy all assemblies to the WebCentral Installation Directory**
   Copy all files that you prepared to the installation directory. This directory depends on the name of your solution, for example, you need to copy these files to the directory

   C:\inetpub\wwwroot\WebCentral\Modules\Primavera.KnowledgeBase\

   if your solution's is name is Primavera.KnowledgeBase.

4) **Copy the WebUI assembly to the bin folder**
   All of the system's module's WebUI assemblies need to be located in the **\bin** folder below the WebCentral's installation directory. *Move* the WebUI assembly from your modules folder to the bin directory.

5) **Register the assemblies in the WebCentral Database**
   Use the tool eModulesUpdate, located in the **\Tools** directory, to update the register of components in the WebCentral Database. This step always should be done, but explicitly needs to be done whenever the version of a module changes.

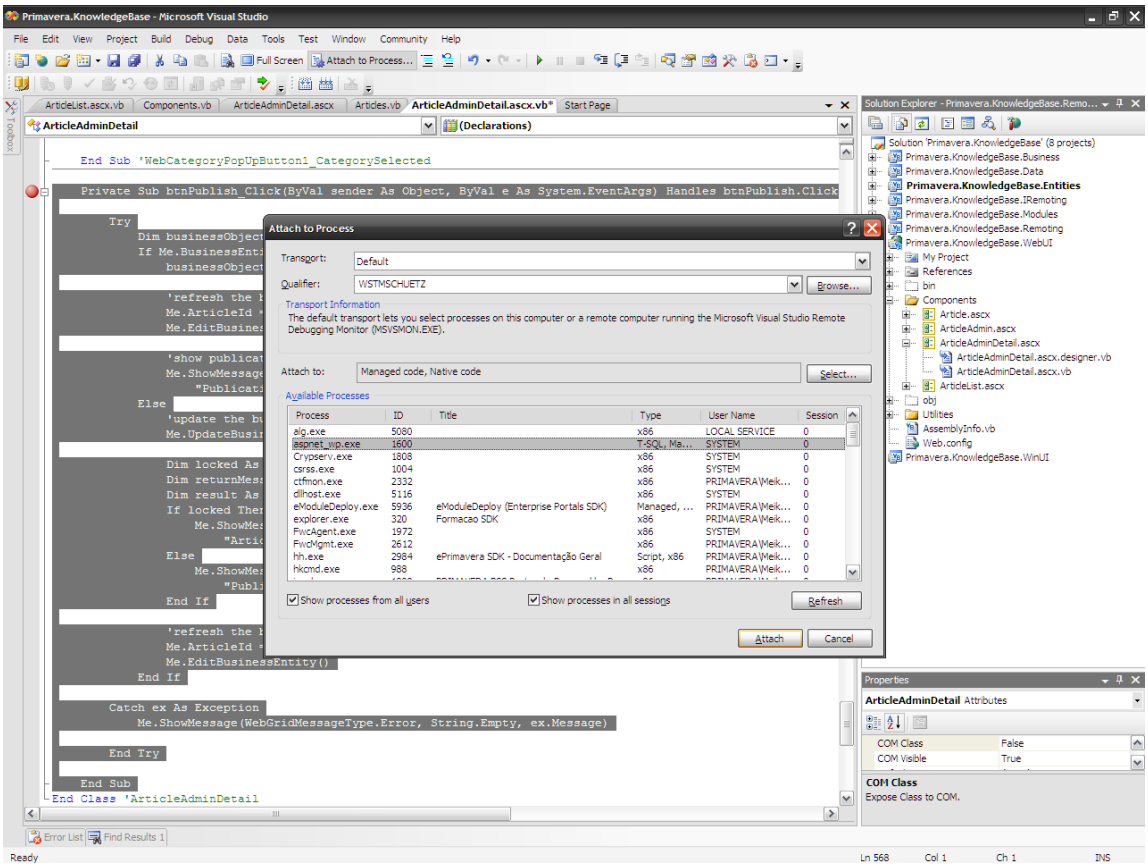
6) **Start the Internet Information Services**
   At last, do not forget to start the website again.

How can I debug my module?

Debugging is a normal process during development and helps to find problems you never might find without. Customized modules are no single web –or windows application that you could debug easily, so you should ask, how can I debug my module?

The answer is: Depends. It depends on what kind of code you are interested. Code that is executed on the client, or code that is executed on the server? Remember, all Windows Forms dialogs are executed in the client's machine!

Most of the time you might need to debug ASP.NET code in your WebUI, Business or Data layer. Just make sure that you compiled and deployed your module in DEBUG mode. Than, before you access any of you components, use the Visual Studio IDE to attach the debugger to the aspnet process.

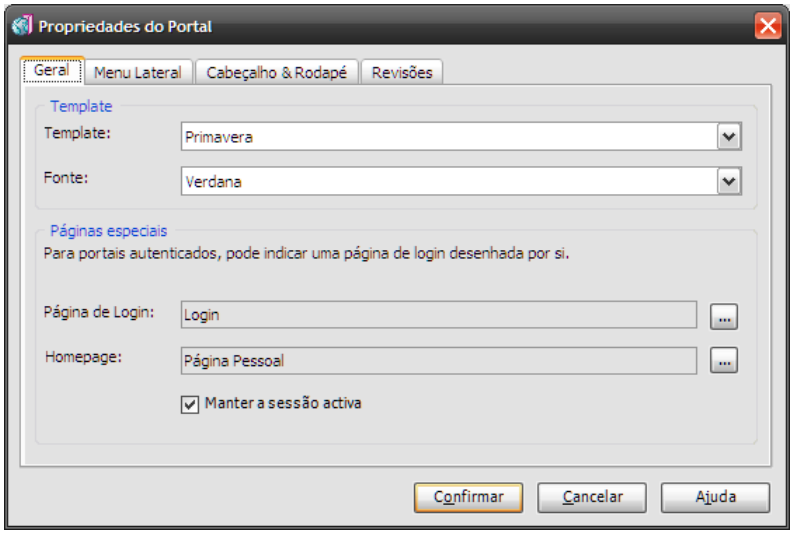


If you need to debug client code, such as in Windows Forms Dialogs (properybag editors), you simply the Visual Studio Debugger to the host, which normally is the Site Administration application or the currently used browser (Internet Explorer).

Once you have attached the Visual Studio Debugger to the right process, all debugging functionality like breakpoints, watch and immediate window are available.

How do I change the portal's template?

PRIMAVERA WebCentral provides you with a feature to change the visual experience using web templates. A web template is a set of Cascading Style Sheets (CSS) and images stored in the WebTemplates folder of the PRIMAVERA WebCentral installation directory. The standard setup already comes along with some web templates, which you can use for your portals. To change the web template used by a portal, simply launch the Site Administration utility and open the Portal Designer for the respective portal. Open up the portals properties and you will be able to choose one of the existing templates for your portal:



The portal will appear with a new look after checking and publishing in the modified portals properties.

How do I create a new layout template?

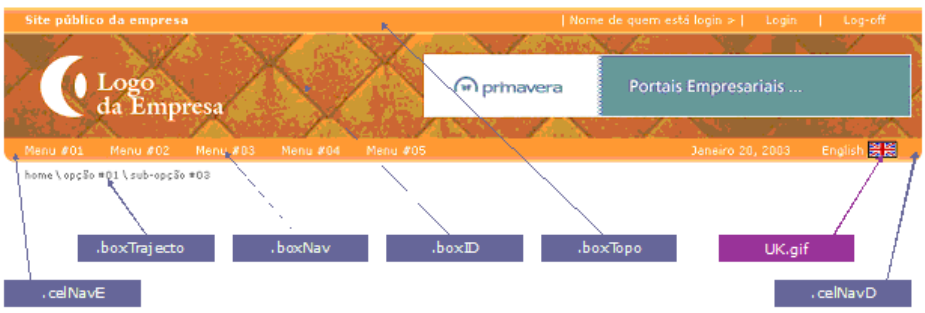
If you feel comfortable with CSS there's nothing that could keep you from customizing one of the available web templates to your own needs. Just take one of the existing templates, copy it to a new web template directory and start modifying the CSS as you wish. The following chapter explains the various styles and where they are used.

Portal Header

The portal header is made of four tables, each one having a different style. These four tables separate different kind of information and make it possible to adjust the visual representation:

- Top (name of the portal and login)
- Corporate Identity Banner
- Navigation (drop-down menu, date and culture)
- Breadcrumb (Navigation Path)

The corporate identity banner can be chosen by the site administrator during the portal's design process, so there's no need for a manual intervention in the CSS file for changing the banner. However the banner should not exceed a maximum height of 80 pixels.



Portal Footer

The portal footer is made of two tables, where the upper one inherits the style from the portals header and the bottom table is made up of following styles:

Text

To distinguish between various types of information, and their respective importance to the user, several styles have been defined for being used:

- Title
- Subtitle
- Headline Title
- Normal text

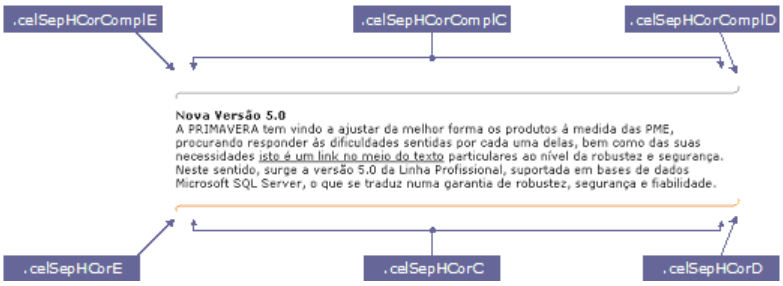
Links

There exist four styles for styling hyperlinks:

- **.linkTopo**, mainly used for the portal header)
- **.linkMenu**, mainly used for the portals menu
- **.linkTxt**, mainly used in other text and in headline section
- **.linkDest**, mainly used in lists with headlines

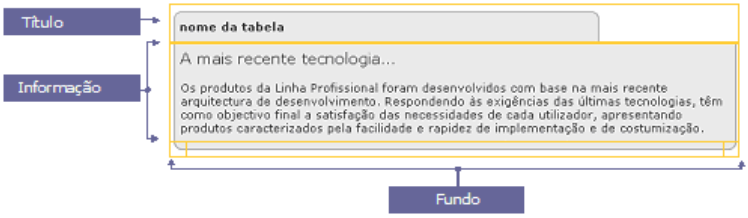
Separators

One way to group and prioritize information is to encapsulate it into a separator. One table, with three columns are needed to build this feature.



Headline sections

A headline sections are composed of 3 tables (title, information and bottom).

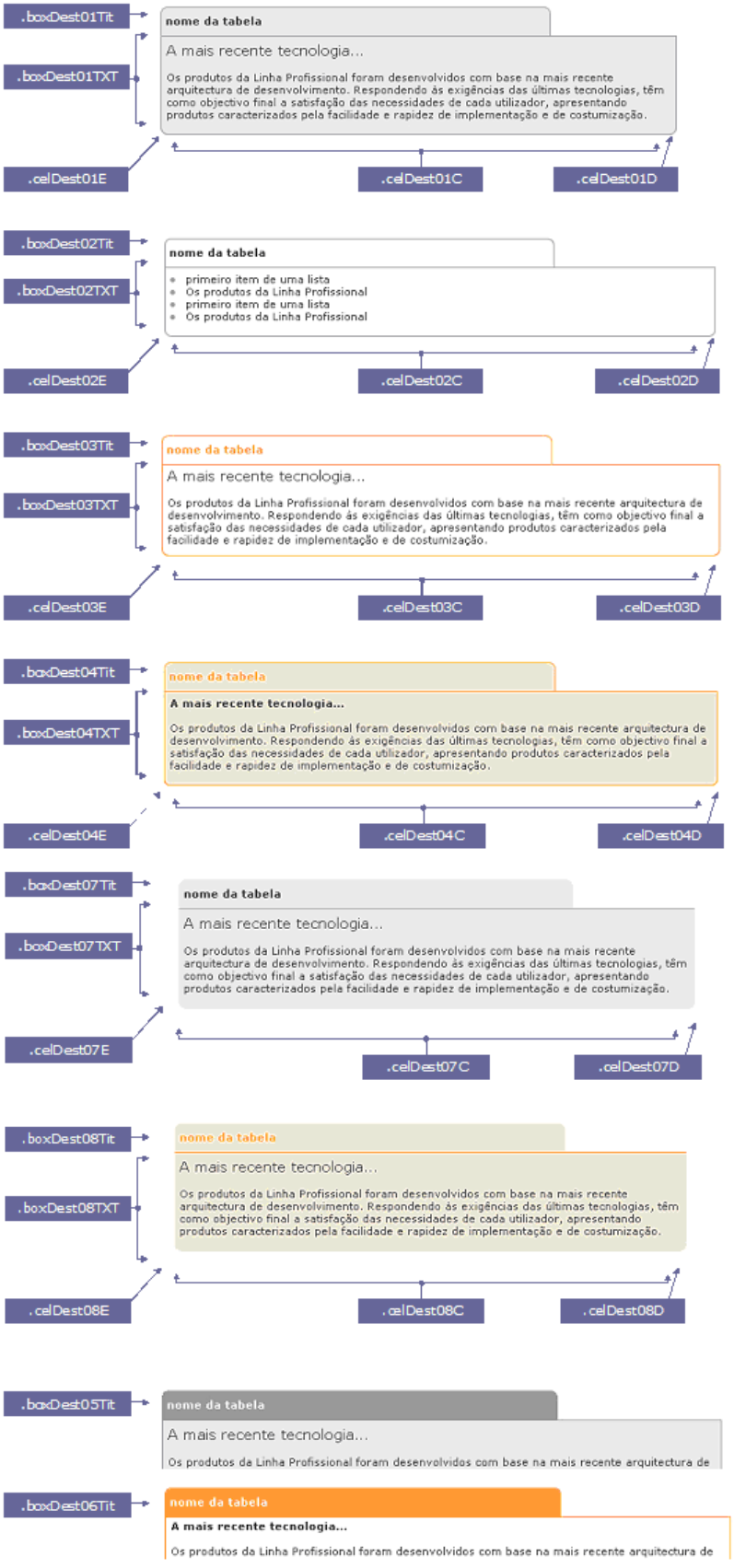


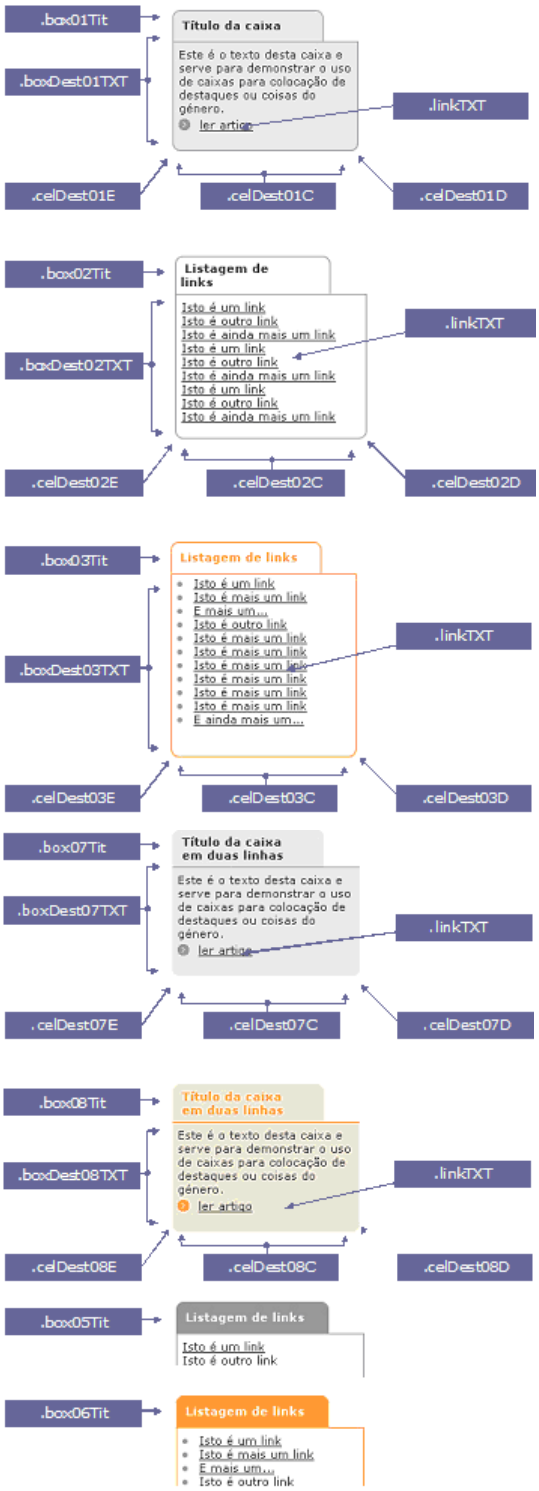
The rule that defines how the various parts fit together is the selected colour. This way, the colour chosen for the headline section also defines the colour for the horizontal and vertical separation lines, the arrow indicator for the links as well as for the buttons, if this applies.

PRIMAVERA WebCentral already comes along with a great variety of headline section styles, which also allow to be combined, and therefore provide a wide range of possibilities.

Attention:

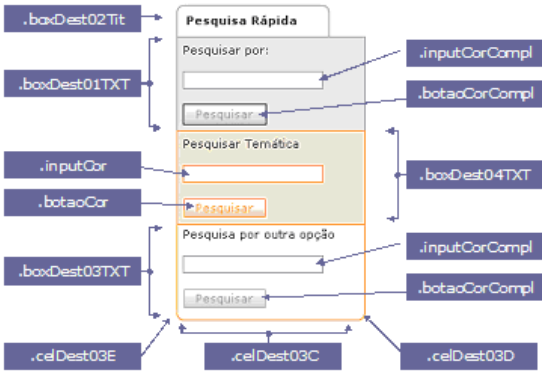
All three tables used to create a highlight box do have three columns. Each of these three columns has style identifier associated, which starts with **.cel** and ends with a key for which column the style applied (**E**=left, **C**=center, **D**=right) – example **.celDest04C** is a style used for the center column of highlight box style 04.


.boxDest01Tit
nome da tabela
.boxDest01TXT
A mais recente tecnologia...
Os produtos da Linha Profissional foram desenvolvidos com base na mais recente arquitectura de desenvolvimento. Respondendo ás exigências das últimas tecnologias, têm como objectivo final a satisfação das necessidades de cada utilizador, apresentando produtos caracterizados pela facilidade e rapidez de implementação e de costumização.
.celDest01E
.celDest01C
.celDest01D
.boxDest02Tit
nome da tabela
.boxDest02TXT
primeiro item de uma lista
Os produtos da Linha Profissional
primeiro item de uma lista
Os produtos da Linha Profissional
.celDest02E
.celDest02C
.celDest02D
.boxDest03Tit
nome da tabela
.boxDest03TXT
A mais recente tecnologia...
Os produtos da Linha Profissional foram desenvolvidos com base na mais recente arquitectura de desenvolvimento. Respondendo ás exigências das últimas tecnologias, têm como objectivo final a satisfação das necessidades de cada utilizador, apresentando produtos caracterizados pela facilidade e rapidez de implementação e de costumização.
.celDest03E
.celDest03C
.celDest03D
.boxDest04Tit
nome da tabela
.boxDest04TXT
A mais recente tecnologia...
Os produtos da Linha Profissional foram desenvolvidos com base na mais recente arquitectura de desenvolvimento. Respondendo ás exigências das últimas tecnologias, têm como objectivo final a satisfação das necessidades de cada utilizador, apresentando produtos caracterizados pela facilidade e rapidez de implementação e de costumização.
.celDest04E
.celDest04C
.celDest04D
.boxDest07Tit
nome da tabela
.boxDest07TXT
A mais recente tecnologia...
Os produtos da Linha Profissional foram desenvolvidos com base na mais recente arquitectura de desenvolvimento. Respondendo ás exigências das últimas tecnologias, têm como objectivo final a satisfação das necessidades de cada utilizador, apresentando produtos caracterizados pela facilidade e rapidez de implementação e de costumização.
.celDest07E
.celDest07C
.celDest07D
.boxDest08Tit
nome da tabela
.boxDest08TXT
A mais recente tecnologia...
Os produtos da Linha Profissional foram desenvolvidos com base na mais recente arquitectura de desenvolvimento. Respondendo ás exigências das últimas tecnologias, têm como objectivo final a satisfação das necessidades de cada utilizador, apresentando produtos caracterizados pela facilidade e rapidez de implementação e de costumização.
.celDest08E
.celDest08C
.celDest08D
.boxDest05Tit
nome da tabela
A mais recente tecnologia...
Os produtos da Linha Profissional foram desenvolvidos com base na mais recente arquitectura de
.boxDest06Tit
nome da tabela
A mais recente tecnologia...
Os produtos da Linha Profissional foram desenvolvidos com base na mais recente arquitectura de


.box01Tit
Título da caixa
Este é o texto desta caixa e serve para demonstrar o uso de caixas para colocação de destaques ou coisas do género.
ler artigo
.boxDest01TXT
.linkTXT
.celDest01E
.celDest01C
.celDest01D
.box02Tit
Listagem de links
Isto é um link
Isto é outro link
Isto é ainda mais um link
Isto é um link
Isto é outro link
Isto é ainda mais um link
Isto é um link
Isto é outro link
Isto é ainda mais um link
.boxDest02TXT
.linkTXT
.celDest02E
.celDest02C
.celDest02D
.box03Tit
Listagem de links
Isto é um link
Isto é mais um link
E mais um...
Isto é outro link
Isto é mais um link
Isto é mais um link
Isto é mais um link
Isto é mais um link
Isto é mais um link
Isto é mais um link
E ainda mais um...
.boxDest03TXT
.linkTXT
.celDest03E
.celDest03C
.celDest03D
.box07Tit
Título da caixa em duas linhas
Este é o texto desta caixa e serve para demonstrar o uso de caixas para colocação de destaques ou coisas do género.
ler artigo
.boxDest07TXT
.linkTXT
.celDest07E
.celDest07C
.celDest07D
.box08Tit
Título da caixa em duas linhas
Este é o texto desta caixa e serve para demonstrar o uso de caixas para colocação de destaques ou coisas do género.
ler artigo
.boxDest08TXT
.linkTXT
.celDest08E
.celDest08C
.celDest08D
.box05Tit
Listagem de links
Isto é um link
Isto é outro link
.box06Tit
Listagem de links
Isto é um link
Isto é mais um link
E mais um...
Isto é outro link

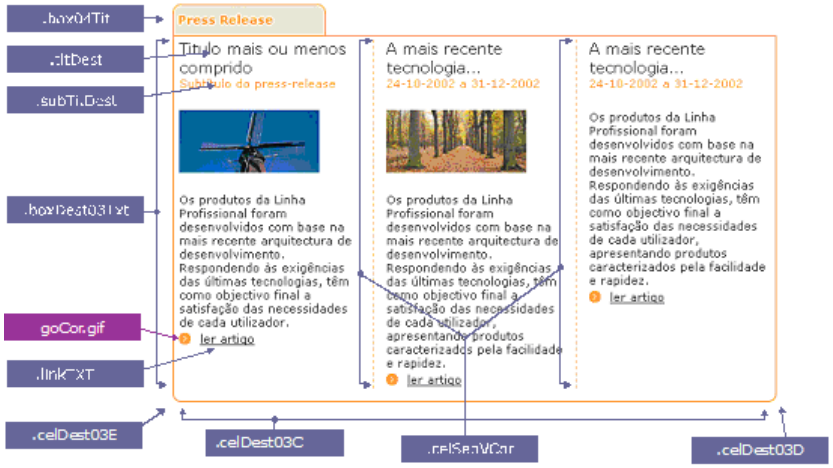
Search Sections

The following model will illustrate the use of styles in the search section.



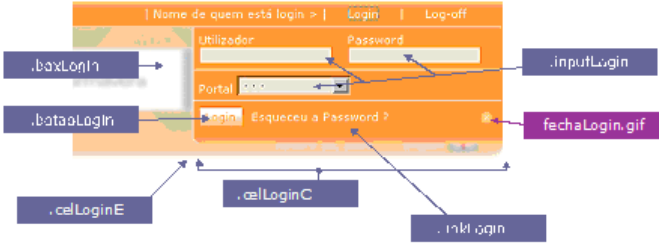
Headlines

The following model will illustrate the use of styles in the headlines section.



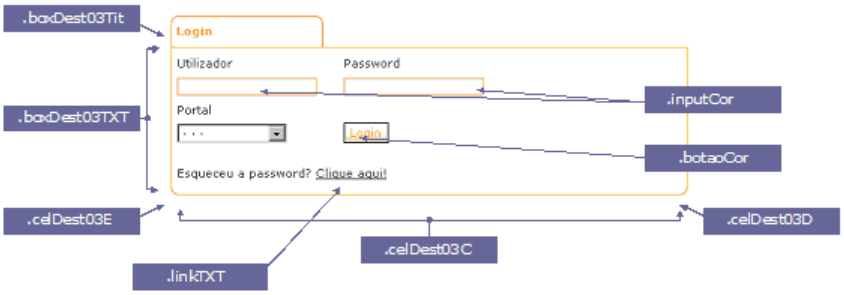
Portal Login

The login interface is composed of four tables located in a <DIV> element at the right corner 24 pixel below the top. The first three tables are attributed to the **boxLogin** style where as the bottom table is divided into two columns. The left column contains a rounded corner (**loginE.gif**) and the right column is attributed to the **celFundoLogin** style.

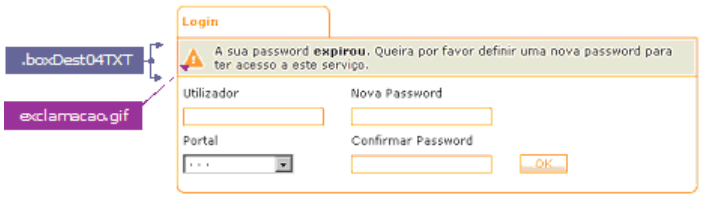


Portal Login Page

The following model will illustrate the use of styles in the login page.

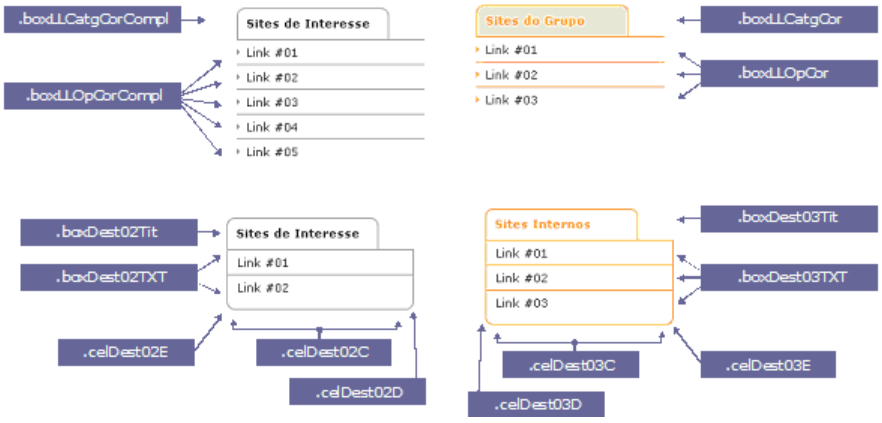


The following image illustrates the styles used in a login page showing a warning.



Link List

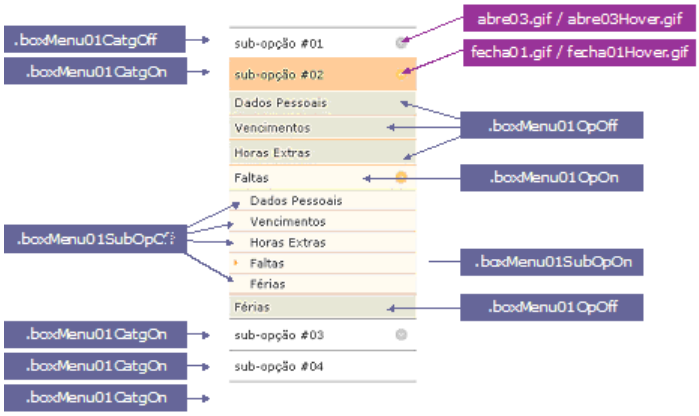
The following image illustrates the usage of styles in the link list component. The style used can be modified in the components properties.



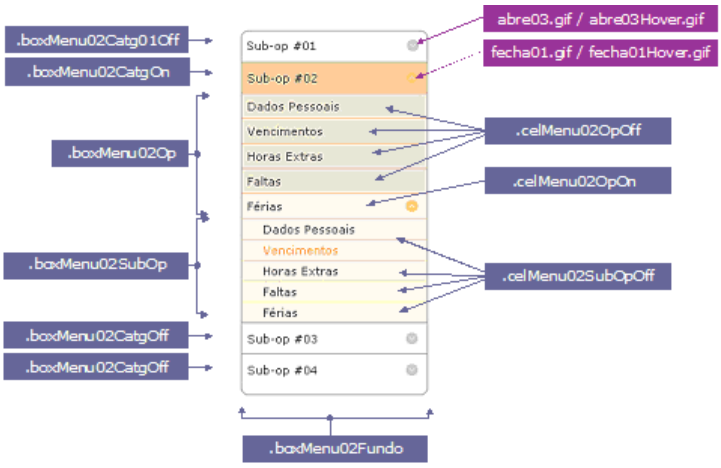
Menu

The PRIMAVERA WebCentral Platform provides two types of menu visualization's. The flat menu style and the rounded menu style – the portals menu style can be selected in the portals properties dialog. The following two images illustrate the usage of styles in the two menu types.

Flat Menu Style

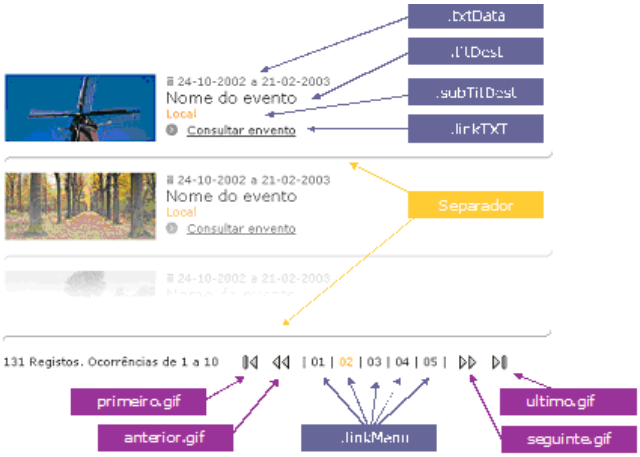


Rounded Menu Style



Content Lists

The following image will illustrate the usage of styles in the content list controls.



Content Options

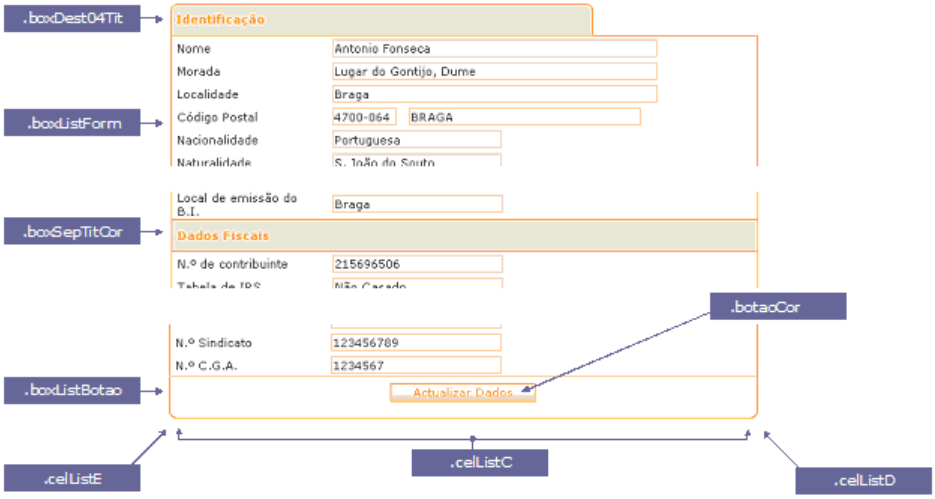
The following image will illustrate the styles used in the content options pane.

Content Headlines

Placed in one of the portal's side columns, this type of component can suggest the user important headlines without the need to use search. The following image explains the style usage of this component.

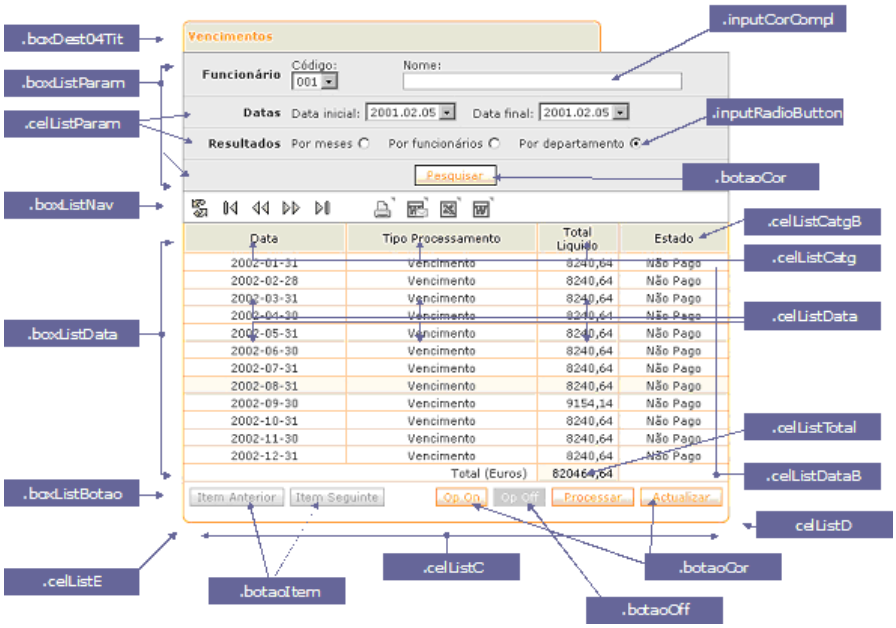
Forms

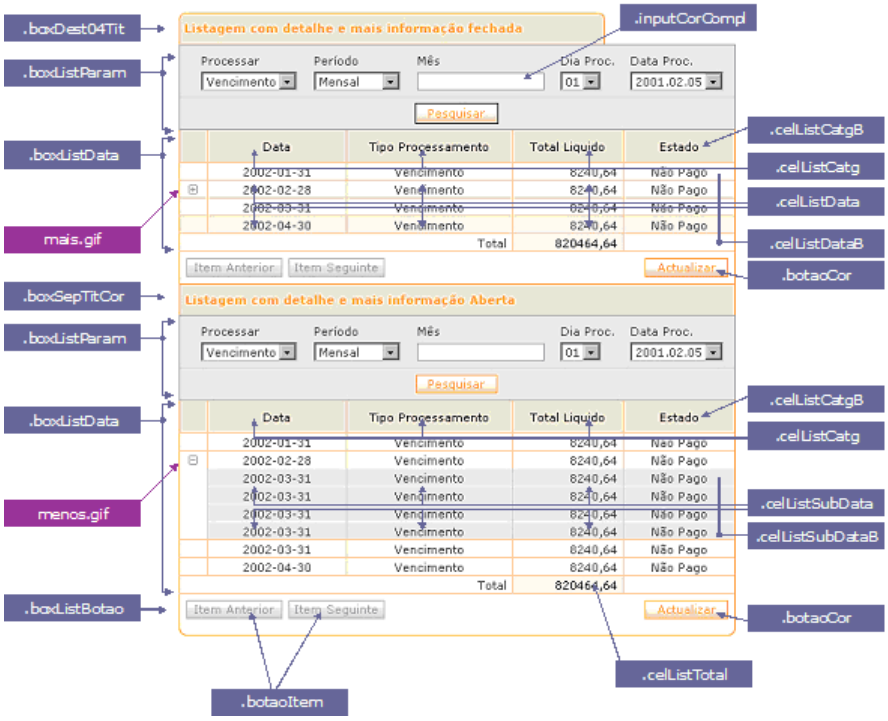
The following image illustrates the usage of styles in the form's component:



Grids

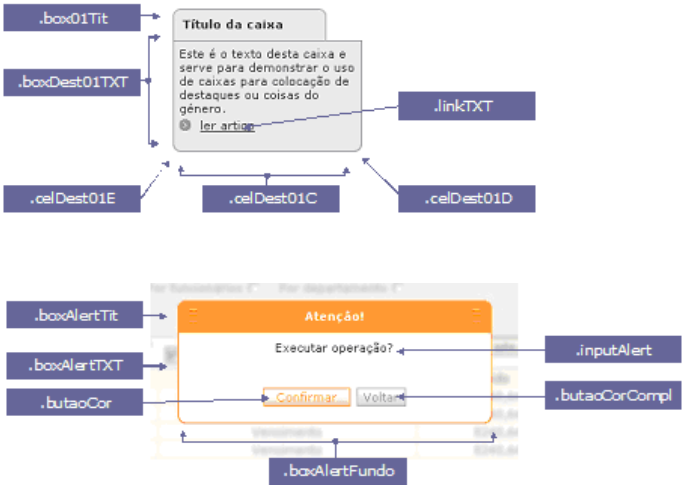
The following two images illustrate the usage of styles in the grid views.





Sections

The following two images illustrate the usage of styles in the information sections.



Appendix

Appendix 1 – Business Entity Article

```
Imports Primavera.Platform.Engine.Entities

<Serializable(), BusinessEntityAttribute("KNB_Articles", "CE1CA692-1864-4806-ACC5-3E2355590B15")> _
Public Class Article
    Inherits BusinessEntityBase

#Region "Private Variables"

    Private m_date As DateTime
    Private m_author As String
    Private m_title As String
    Private m_summary As String
    Private m_body As String
    Private m_image As String

#End Region

#Region "Public Properties"

    <BusinessEntityField("Date")> _
    Public Property [Date]() As DateTime
        Get
            Return m_date
        End Get
        Set(ByVal Value As DateTime)
            m_date = Value
        End Set
    End Property

    <BusinessEntityField("Author")> _
    Public Property Author() As String
        Get
            Return m_author
        End Get
        Set(ByVal Value As String)
            m_author = Value
        End Set
    End Property

    <BusinessEntityField("Title")> _
    Public Property Title() As String
        Get
            Return m_title
        End Get
        Set(ByVal Value As String)
            m_title = Value
        End Set
    End Property

    <BusinessEntityField("Summary")> _
```

```
    Public Property Summary() As String
        Get
            Return m_summary
        End Get
        Set(ByVal Value As String)
            m_summary = Value
        End Set
    End Property

    <BusinessEntityField("Body")> _
    Public Property Body() As String
        Get
            Return m_body
        End Get
        Set(ByVal Value As String)
            m_body = Value
        End Set
    End Property

    <BusinessEntityField("Image")> _
    Public Property Image() As String
        Get
            Return m_image
        End Get
        Set(ByVal Value As String)
            m_image = Value
        End Set
    End Property

#End Region

End Class
```

Appendix 2 – Business Service

```
Imports Primavera.Platform
Imports Primavera.Platform.Engine.Entities
Imports Primavera.Platform.Engine.Business

<BusinessServiceAttribute(GetType(Data.Articles))> _
Public Class Articles
    Inherits BusinessServiceBase

#Region "Editing"

    Public Function Edit(ByVal Context As EngineContext, ByVal Id As Guid) As Entities.Article
        Return CType(BaseEdit(Context, Id), Entities.Article)
    End Function

    Public Function EditField(ByVal Context As EngineContext, ByVal Id As Guid, ByVal Field As
String) As Object
        Return BaseEditField(Context, Id, Field)
    End Function

    Public Function EditFields(ByVal Context As EngineContext, ByVal Id As Guid, ByVal ParamArray
Fields() As String) As Fields
        Return BaseEditFields(Context, Id, Fields)
    End Function

#End Region

#Region "Updating"

    Public Sub Update(ByVal Context As EngineContext, ByVal Article As Entities.Article)
        BaseUpdate(Context, CType(Article, BusinessEntityBase))
    End Sub

    Public Sub UpdateField(ByVal Context As EngineContext, ByVal Id As Guid, ByVal Field As String,
ByVal Value As Object)
        BaseUpdateField(Context, Id, Field, Value)
    End Sub

    Public Sub UpdateFields(ByVal Context As EngineContext, ByVal Id As Guid, ByRef Fields As
Fields)
        BaseUpdateFields(Context, Id, Fields)
    End Sub

#End Region

#Region "Existing"

    Public Function Exists(ByVal Context As EngineContext, ByVal Id As Guid) As Boolean
        Return BaseExists(Context, Id)
    End Function

#End Region

#Region "Removing"
```

```
    Public Sub Remove(ByVal Context As EngineContext, ByVal Id As Guid)
        BaseRemove(Context, Id)
    End Sub

#End Region

#Region "Locking"

    Public Sub LockUI(ByVal Context As EngineContext, ByVal Id As Guid)
        BaseLockUI(Context, Id)
    End Sub

    Public Sub UnlockUI(ByVal Context As EngineContext, ByVal Id As Guid)
        BaseUnlockUI(Context, Id)
    End Sub

#End Region

#Region "Validations"

    Public Overloads Function ValidateUpdate(ByVal Context As EngineContext, ByVal Article As
Entities.Article, ByRef OutMessage As String, Optional ByVal FieldsToValidate As String = "*") As
Boolean
        Return BaseValidateUpdate(Context, CType(Article, BusinessEntityBase), OutMessage,
FieldsToValidate)
    End Function

    Public Overloads Function ValidateRemove(ByVal Context As EngineContext, ByVal Id As Guid,
ByRef OutMessage As String) As Boolean
        Return BaseValidateRemove(Context, Id, OutMessage)
    End Function

#End Region

#Region "Cloning"

    Protected Overrides Function BaseClone(ByVal Context As EngineContext, ByVal Id As Guid) As
Guid
INICIO:
        Try
            Context.BeginTransaction()
            Dim cloneId As Guid = MyBase.BaseClone(Context, Id)

            'TODO: clonar classes de detalhe que proventura existam.

            Context.CommitTransaction()
            Return cloneId

        Catch ex As Exception
            Context.RollBackTransaction()
            If Context.VerifyLockException(ex) Then GoTo INICIO
            Throw ex

        End Try
    End Function
```

```
    Public Function Clone(ByVal Context As EngineContext, ByVal Id As Guid) As Guid
        Return BaseClone(Context, Id)
    End Function

#End Region

#Region "Security"

    Protected Overloads Overrides Function BaseVerifyEditPermission(ByVal Context As EngineContext,
ByVal ID As Guid, ByRef OutMessage As String) As Boolean
        Return MyBase.BaseVerifyEditPermission(Context, ID, OutMessage)
    End Function

    Protected Overloads Overrides Function BaseVerifyUpdatePermission(ByVal Context As
EngineContext, ByVal Entity As BusinessEntityBase, ByRef OutMessage As String, Optional ByVal
FieldsToValidate As String = "*") As Boolean
        Return MyBase.BaseVerifyUpdatePermission(Context, Entity, OutMessage, FieldsToValidate)
    End Function

    Protected Overrides Function BaseVerifyNewPermission(ByVal Context As EngineContext, ByVal
Entity As BusinessEntityBase, ByRef OutMessage As String) As Boolean
        Return MyBase.BaseVerifyNewPermission(Context, Entity, OutMessage)
    End Function

    Protected Overloads Overrides Function BaseVerifyRemovePermission(ByVal Context As
EngineContext, ByVal ID As Guid, ByRef OutMessage As String) As Boolean
        Return MyBase.BaseVerifyRemovePermission(Context, ID, OutMessage)
    End Function

    Public Overloads Function VerifyEditPermission(ByVal Context As EngineContext, ByVal ID As
Guid, ByRef OutMessage As String) As Boolean
        Return BaseVerifyEditPermission(Context, ID, OutMessage)
    End Function

    Public Overloads Function VerifyNewPermission(ByVal Context As EngineContext, ByVal Article As
Entities.Article, ByRef OutMessage As String) As Boolean
        Return BaseVerifyNewPermission(Context, CType(Article, BusinessEntityBase), OutMessage)
    End Function

    Public Overloads Function VerifyRemovePermission(ByVal Context As EngineContext, ByVal ID As
Guid, ByVal OutMessage As String) As Boolean
        Return BaseVerifyRemovePermission(Context, ID, OutMessage)
    End Function

    Public Overloads Function VerifyUpdatePermission(ByVal Context As EngineContext, ByVal Article
As Entities.Article, ByVal OutMessage As String) As Boolean
        Return BaseVerifyUpdatePermission(Context, CType(Article, BusinessEntityBase), OutMessage)
    End Function

#End Region


End Class
```

Appendix 3 – Article Web User Control (HTML)

```
<%@ Control Language="vb" AutoEventWireup="false" Codebehind="Article.ascx.vb"
Inherits="Primavera.KnowledgeBase.WebUI.Article"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<table>
     <tr>
          <td rowspan="2"><asp:Literal id="litImage" runat="server"></asp:Literal></td>
          <td><asp:Label id="lblTitle" runat="server"
CssClass="Title"></asp:Label></td>
     </tr>
     <tr>
          <td><asp:Label id="lblSummary" runat="server"
CssClass="SubTitle"></asp:Label></td>
     </tr>
     <tr>
          <td colspan="2"><asp:Label id="lblBody" runat="server"></asp:Label></td>
     </tr>
     <tr>
          <td colspan="2"><b><asp:Label id="lblAuthor"
runat="server"></asp:Label></b></td>
     </tr>
</table>
```

Appendix 4 – Article Web User Control (Code behind)

```
Public Class Article
    Inherits Primavera.Platform.WebUI.WebComponentBase

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

    End Sub
    Protected WithEvents litImage As System.Web.UI.WebControls.Literal
    Protected WithEvents lblTitle As System.Web.UI.WebControls.Label
    Protected WithEvents lblSummary As System.Web.UI.WebControls.Label
    Protected WithEvents lblBody As System.Web.UI.WebControls.Label
    Protected WithEvents lblAuthor As System.Web.UI.WebControls.Label

    'NOTE: The following placeholder declaration is required by the Web Form Designer.
    'Do not delete or move it.
    Private designerPlaceholderDeclaration As System.Object

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private m_article As Entities.Article

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        If Not Request("ArticleID") Is Nothing Then
            Dim ArticleID As Guid = New Guid(Request("ArticleID"))
            LoadArticle(ArticleID)
            ShowArticle()
        Else
            Me.Visible = False
        End If
    End Sub

    Private Sub LoadArticle(ByVal articleId As Guid)
        m_article = Business.Proxy.Articles.Edit(WebContext.User.EngineContext,
articleId)
    End Sub
```

```
    Private Sub ShowArticle()
        With m_article
            litImage.Text = .Image
            lblAuthor.Text = .Author
            lblTitle.Text = .Title
            lblSummary.Text = .Summary
            lblBody.Text = .Body
        End With
    End Sub
End Class
```

Appendix 5 – Article List Web User Control (HTML)

```
<%@ Control Language="vb" AutoEventWireup="false" Codebehind="ArticleList.ascx.vb"
Inherits="Primavera.KnowledgeBase.WebUI.ArticleList"
TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<asp:DataList id="lstArtigos" runat="server">
<ItemTemplate>
<table>
<tr >
<td><%# DataBinder.Eval(Container.DataItem, "Image")%></td>
     <td>
          <table>
          <tr>
<td><a href = <%# GetDetailPageUrl(DataBinder.Eval(Container.DataItem, "ID"))%>
class="LinkText">
<%# DataBinder.Eval(Container.DataItem, "Title")%></a></td>
          </tr>
          <tr>
              <td><%# DataBinder.Eval(Container.DataItem, "Author")%></td>
          </tr>
          <tr>
              <td><%# DataBinder.Eval(Container.DataItem, "Summary")%></td>
          </tr>
          <tr>
              <td><%# DataBinder.Eval(Container.DataItem, "Date")%></td>
          </tr>
          </table>
     </td>
</tr>
</table>
</ItemTemplate>
</asp:DataList>
```

Appendix 6 – Article List Web User Control (Code behind)

```
Public Class ArticleList
    Inherits Primavera.Platform.WebUI.WebComponentBase

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

    End Sub

    'NOTE: The following placeholder declaration is required by the Web Form Designer.
    'Do not delete or move it.
    Private designerPlaceholderDeclaration As System.Object

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
        LoadArticles()
    End Sub

    Private Sub LoadArticles()
        Dim strSql As String = "SELECT * FROM KNB_Articles ORDER BY [date]"
        Dim tblArtigos As DataTable =
Primavera.Platform.Engine.Business.List.List(WebContext.User.EngineContext, strSql)
        lstArtigos.DataSource = tblArtigos
        lstArtigos.DataBind()
    End Sub

    Public Function GetDetailPageUrl(ByVal articleId As Guid) As String
        Dim guidCompArticleDetail As Guid = New Guid("B6B669F2-35B2-43a1-A03D-
7325EEB2B082")
        Return String.Format("ComponentRender.aspx?ComponentID={0}&ArticleID={1}",
guidCompArticleDetail, articleId)
    End Function
End Class
```

Appendix 7 – SQL Script for the KnowledgeBase

```
if exists (select * from dbo.sysobjects
     where id = object_id(N'[dbo].[KNB_Articles]') and OBJECTPROPERTY(id,
N'IsUserTable') = 1)
drop table [dbo].[KNB_Articles]
GO

CREATE TABLE [dbo].[KNB_Articles] (
     [ID] [uniqueidentifier] NOT NULL,
     [Date] [datetime] NULL,
     [Author] [nvarchar] (100) NULL,
     [Title] [nvarchar] (250)  NULL,
     [Summary] [nvarchar] (500),
     [Body] [nvarchar] (500)  NULL,
     [Image] [nvarchar] (250) NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[KNB_Articles] WITH NOCHECK ADD
     CONSTRAINT [PK_KNB_Articles] PRIMARY KEY  CLUSTERED
     (
          [ID]
     )  ON [PRIMARY]
GO

INSERT INTO [dbo].[KNB_Articles] (ID,[Date],Author,Title,Summary,Body,[Image])
     VALUES (NEWID(),GETDATE(),'Jim Button','Desenvolvendo um Módulo','This article
describes ...',
     'Para desenvolver um Módulo WEBC é imprescindivel começar por uma leitura
adequada...',
     '<img src="UserFiles/Images/003.bmp" >')
INSERT INTO [dbo].[KNB_Articles] (ID,[Date],Author,Title,Summary,Body,[Image])
     VALUES (NEWID(),GETDATE(),'Laura Jones','Integração','Discutindo a integração
...',
     'Este assunto é susceptivel de levantar alguma polemica. Apesar de existir
bastante documentação sobre este tema ...',
'<img src="UserFiles/Images/002.bmp" >')
```

Appendix 8 – Article Administration Details Control (HTML)

```
<%@ Control Language="vb" AutoEventWireup="false"
CodeBehind="ArticleAdminDetail.ascx.vb"
Inherits="Primavera.KnowledgeBase.WebUI.ArticleAdminDetail" %>
<%@ Register TagPrefix="WebC" Namespace="Infragistics.WebUI.WebSchedule"
Assembly="Infragistics2.WebUI.WebDateChooser.v6.2, Version=6.2.20062.34,
Culture=neutral, PublicKeyToken=7dd5c3163f2cd0cb" %>
<%@ Register TagPrefix="WebC" TagName="WebImageSelection"
Src="../Primavera.Platform/WebImageSelection.ascx" %>
<%@ Register TagPrefix="WebC" TagName="WebImageSelectionPopUpButton"
Src="../Primavera.Platform/WebImageSelectionPopUpButton.ascx" %>
<!--
    The following script will include the necessary java script to handle a web based
    HTML editor, which we'll use in this example for composing the article
-->
<SCRIPT language="javascript">
<!--
     _editor_url = "<%=GetCurrentPortalURL(Request)%>";
     _editor_template = "<%="WebTemplates/" & WebContext.Portal.Template & "/Styles/" &
System.Enum.GetName(GetType(Primavera.Platform.Security.Entities.PortalConfig.FontStyle
s), WebContext.Portal.Font) & ".css"%>";
     var win_ie_ver = parseFloat(navigator.appVersion.split("MSIE")[1]);
     if (navigator.userAgent.indexOf('Mac')        >= 0) { win_ie_ver = 0; }
     if (navigator.userAgent.indexOf('Windows CE') >= 0) { win_ie_ver = 0; }
     if (navigator.userAgent.indexOf('Opera')      >= 0) { win_ie_ver = 0; }
     if (win_ie_ver >= 5.5) {
          document.write('<scr' + 'ipt src="' +_editor_url+
'/SystemFiles/Scripts/editor_contentmanagement.js"');
          document.write(' language="Javascript1.2"></scr' + 'ipt>');
     } else { document.write('<scr'+'ipt>function editor_generate() { return false;
}</scr'+'ipt>'); }
// -->
</SCRIPT>
<asp:Panel ID="panelArticleDetails" runat="server" Visible="True">
    <!--
        The panel contains all controls which are necessary to insert/edit a
new/existing
        item; it will demonstrate some special controls, such as the Infragistics
DateTime
        picker as well as the HTML editor, for composing the article body. This panel
will
        be hidden whenever the user clicked the image button. To select an image for
the
        message this panel will be hidden and the image selection panel (see below)
will be
        shown.
    -->
```

```
    <table>
    <tr>
        <!-- Articles Publishing Date (Infragistics DateTime Picker) -->
        <td class="width_s alignRight">Date</td>
        <td class="width_s">
            <WebC:webdatechooser id="txtDate" runat="server"></WebC:webdatechooser>
        </td>
    </tr>
    <tr>
        <!-- Articles Autor Name -->
        <td class="width_s alignRight">Author</td>
        <td class="width_xl">
         <asp:textbox id="txtAutor" onblur="this.className='data_input width_xl';"
             onfocus="this.className='data_input_over width_xl';"
                 runat="server" CssClass="data_input width_xl" MaxLength="50">
             </asp:textbox>
        </td>
    </tr>
    <tr>
        <!-- Articles Image (PRIMAVERA WebCentral Platform WebImageSelection) -->
        <td class="width_s alignRight">Image</td>
        <td class="width_xl">
         <asp:textbox id="txtImage" onblur="this.className='data_input width_l';"
             onfocus="this.className='data_input_over width_l';"
                 runat="server" CssClass="data_input width_l" MaxLength="50">
             </asp:textbox>
          <WebC:WebImageSelectionPopUpButton id="WebImageSelectionPopUpButton1"
runat="server"></WebC:WebImageSelectionPopUpButton>
        </td>
    </tr>
    <tr>
        <!-- Articles Title -->
        <td class="width_s alignRight">Title</td>
        <td class="width_xl">
         <asp:textbox id="txtTitle" onblur="this.className='data_input width_xl';"
             onfocus="this.className='data_input_over width_xl';"
                 runat="server" CssClass="data_input width_xl" MaxLength="50">
             </asp:textbox>
        </td>
    </tr>
    <tr>
        <!-- Articles Message Body (HTML editor) -->
        <td class="width_s alignRight">Article</td>
        <td class="width_xl">
            <asp:textbox id="txtArticle" onblur="this.className='data_input width_xl';"
             onfocus="this.className='data_input_over width_xl';"
                 runat="server" CssClass="data_input width_xl" MaxLength="50">
             </asp:textbox>
        </td>
    </tr>
```

```
    </table>
    <!-- initialize the HTML editor (control id txtArticle) -->
    <script language="javascript" defer>
        editor_generate('<%=ID2Name(Me.ClientID)%>:txtArticle');
    </script>
    <!-- command button area header -->
     <table cellSpacing="0" cellPadding="0" width="100%" border="0">
     <tr>
          <TD class="filter_top_l"><IMG src="SystemFiles/Images/null.gif"></TD>
          <TD class="filter_top_c"><IMG src="SystemFiles/Images/null.gif"></TD>
          <TD class="filter_top_r"><IMG src="SystemFiles/Images/null.gif"></TD>
     </tr>
     </table>
     <!-- command button area content -->
     <table class="filter" cellSpacing="0" cellPadding="0" width="100%" border="0">
     <tr>
          <td class="filter_l"> </td>
          <td>
              <table class="filter_btns" cellSpacing="0" cellPadding="0" border="0">
                   <tr>
                        <td>
<asp:button id="btnConfirm"
onmouseover="this.className='btn_m_color_over';"
onmouseout="this.className='btn_m_color';"
                             runat="server" CssClass="btn_m_color"
                             Text="Confirm"></asp:button></td>
                        <td>
                             <asp:button id="btnBack"
onmouseover="this.className='btn_m_color_over';"
onmouseout="this.className='btn_m_color';"
                             runat="server" CssClass="btn_m_color"
Text="Back"></asp:button></td>
                   </tr>
              </table>
          </td>
          <td class="filter_r"> </td>
     </tr>
     </table>
     <!-- command button area footer -->
     <table cellSpacing="0" cellPadding="0" width="100%" border="0">
          <tr>
              <td class="filter_bottom_l"><IMG src="SystemFiles/Images/null.gif"></td>
              <td class="filter_bottom_c"><IMG src="SystemFiles/Images/null.gif"></td>
              <td class="filter_bottom_r"><IMG src="SystemFiles/Images/null.gif"></td>
          </tr>
     </table>
</asp:Panel>
<asp:panel id="panelImageSelection" runat="server" Visible="False">
    <!-- image selection control panel; will only be shown when the user clicked on
         the image selection button in the details panel. -->
```

```
    <WebC:webimageselection id="controlImageSelection"
runat="server"></WebC:webimageselection>
</asp:panel>
```

Appendix 9 – Article Administration Details User Control (VB)

```
Public Partial Class ArticleAdminDetail
    Inherits Web.UI.UserControl

    'declare events to signal the parent user control or form that the user clicked
    'on the back button or the confirm button. the parent user control is responsible
    'for reacting the correct way on these events.
    Public Event BackButtonClicked()
    Public Event ArticleSaved()

    'declare user controls referenced from the PRIMAVERA WebCentral Platform
    'here in the code behind file, using the exact variable name as stated in the ascx
    Protected WithEvents WebImageSelectionPopUpButton1
As Primavera.Platform.WebUI.WebImageSelectionPopUpButton
    Protected WithEvents controlImageSelection
As Primavera.Platform.WebUI.WebImageSelection

    'private attributes, which are initialized by the parent user control using the
    'public properties this class exposes
    Private _webContext As Primavera.Platform.WebUI.WebContext
    Private _visualizationMode As VisualizationMode

    '<summary>
    '  This property allows access to vital context information, that are necessary to
    '  implement access to the system's database (such as information about user,
    '  organization, portal, etc). This property needs to be initialized from the
parent
    '  user control.
    '</summary>
    Public Property WebContext() As Primavera.Platform.WebUI.WebContext
        Get
            Return _webContext
        End Get
        Set(ByVal Value As Primavera.Platform.WebUI.WebContext)
            _webContext = Value
        End Set
    End Property

    '<summary>
    '  This property allows access to the edit mode; this control needs to know if we
are
    '  currently editing an existing, or inserting a new record. This property needs to
    '  be initialized from the parent user control.
    '</summary>
    Public Property Mode() As VisualizationMode
        Get
```

```
            Return _visualizationMode
        End Get
        Set(ByVal Value As VisualizationMode)
            _visualizationMode = Value
            Select Case _visualizationMode
                Case VisualizationMode.Insertion
                    CreateBusinessEntity()
                Case VisualizationMode.Edition
                    EditBusinessEntity()
            End Select
            ShowBusinessEntity()
        End Set
    End Property 'Mode

    '<summary>
    '  The parent user control needs to initialize this property whenever the user is
    '  editing an existing record. This property needs to be initialized with the
    '  primary key of the record to be edited.
    '</summary>
    Public Property ArticleId() As Guid
        Get
            Return CType(Session("ArticleId"), System.Guid)
        End Get
        Set(ByVal Value As System.Guid)
            Session("ArticleId") = Value
        End Set
    End Property

    '<summary>
    '  The following property persists an business entity object of the record that is
    '  currently edited. This property is for internal use only.
    '</summary>
    Private Property BusinessEntity() As Entities.Article
        Get
            If Not Session("ArticleBEO") Is Nothing Then
                Return CType(Session("ArticleBEO"), Entities.Article)
            End If
            Return Nothing
        End Get
        Set(ByVal Value As Entities.Article)
            Session("ArticleBEO") = Value
        End Set
    End Property

    '<summary>
    '  The following method will be called in the ASCX file to retrieve an HTML valid
    '  identifier to reference the textbox control used as HTML editor (see also ASCX)
    '</summary>
    Protected Function ID2Name(ByVal ID As String) As String
```

```
        Return System.Text.RegularExpressions.Regex.Replace(ID, "(?<tok>[a-zA-Z0-
9]+)_", "${tok}:")

    End Function 'ID2Name

    '<summary>
    '  The following method will be called to build a correct portal URL, for including
    '  JavaScript files for the HTML editor (see also ASCX)
    '</summary>
    Protected Function GetCurrentPortalURL(ByVal Request As HttpRequest) As String

        Dim currentPortalUrl As String = String.Format("http://{0}/{1}", _
            Request.Url.Host, Request.Url.Segments(1))
        Return currentPortalUrl.TrimEnd("/")

    End Function 'GetCurrentPortalURL

    '<summary>
    '  The following method will cleanup the session object which are used by this
    '  user control. You should call this method whenever the in the session state
    '  persisted objects are no longer necessary.
    '</summary>
    Private Sub ClearSessionObjects()

        Session.Remove("ArticleId")
        Session.Remove("ArticleBEO")

    End Sub 'ClearSessionObjects

    '<summary>
    '  The following method will be called whenever it is necessary to initialize
    '  the user control with values from the currently loaded business entity. This
    '  is basically everytime the case, when a user edits an existing record.
    '</summary>
    Private Sub ShowBusinessEntity()

        txtAutor.Text = Me.BusinessEntity.Author
        txtArticle.Text = Me.BusinessEntity.Body
        txtDate.Value = Me.BusinessEntity.Date
        txtTitle.Text = Me.BusinessEntity.Title
        txtImage.Text = Me.BusinessEntity.Image

        'update image selection control
        WebImageSelectionPopUpButton1.ImageHtml = Me.BusinessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'ShowBusinessEntity

    '<summary>
    '  The following method will be called whenever the user wishes to create
```

```
    '  a new record. This method initializes a new business entity object.
    '</summary>
    Private Sub CreateBusinessEntity()

        'initialize new business entity
        Dim businessEntity As New Entities.Article
        businessEntity.ID = Guid.NewGuid()
        businessEntity.Author = WebContext.User.UserName
        businessEntity.Date = DateTime.Now
        businessEntity.Image = String.Empty
        businessEntity.Title = String.Empty
        businessEntity.Body = String.Empty

        'initialize image button
        WebImageSelectionPopUpButton1.ImageHtml = businessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

        Me.BusinessEntity = businessEntity

    End Sub 'CreateBusinessEntity

    '<summary>
    '  The following method will be called to initialize the internally used business
    '  entity object for being edited. It will load the specified record from database;
    '  the record's primary key needs to be set before.
    '</summary>
    Private Sub EditBusinessEntity()

        Dim businessLayer As New Business.Articles
        Me.BusinessEntity = businessLayer.Edit(WebContext.User.EngineContext,
Me.ArticleId)

        'initialize image button
        WebImageSelectionPopUpButton1.ImageHtml = Me.BusinessEntity.Image
        WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'EditBusinessEntity

    '<summary>
    '  The following method will be called whenever it is necessary to write all
changes
    '  down to database. This method will update the existing record, which was edited,
or
    '  inserts a new record to the database.
    '</summary>
    Private Sub UpdateBusinessEntity()

        Me.BusinessEntity.Author = txtAutor.Text
        Me.BusinessEntity.Title = txtTitle.Text
        Me.BusinessEntity.Body = txtArticle.Text
```

```
        Me.BusinessEntity.Date = txtDate.Value
        Me.BusinessEntity.Image = txtImage.Text

        Dim businessLayer As New Business.Articles
        businessLayer.Update(WebContext.User.EngineContext, Me.BusinessEntity)

    End Sub 'UpdateBusinessEntity

    '<summary>
    '  The following event handler will be called whenever the page loads; place all
    '  initialization code here.
    '</summary>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        Me.WebImageSelectionPopUpButton1.PanelMaster = Me.panelArticleDetails
        Me.WebImageSelectionPopUpButton1.PanelDetail = Me.panelImageSelection
        Me.WebImageSelectionPopUpButton1.WebImageSelectionControl =
Me.controlImageSelection
        If Not Me.IsPostBack Then
Me.WebImageSelectionPopUpButton1.SetConfigurationParameters()

    End Sub 'Page_Load

    '<summary>
    '  The following event handler will be called whenever the user clicked on the
"Back"
    '  button. No changes should be written down to database; the parent will receive
an
    '  event and should hide this details control and show the table filter grid.
    '</summary>
    Private Sub cmdBack_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnBack.Click

        RaiseEvent BackButtonClicked()
        ClearSessionObjects()

    End Sub 'cmdBack_Click

    '<summary>
    '  The following event handler will be called whenever the user clicked on the
"Confirm"
    '  button. All changes should be written down to database; ; the parent will
receive an
    '  event and should hide this details control and show the table filter grid.
    '</summary>
    Private Sub cmdSave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnConfirm.Click

        Me.UpdateBusinessEntity()
```

```
        RaiseEvent ArticleSaved()
        ClearSessionObjects()

    End Sub 'cmdSave_Click

    '<summary>
    '  The following event handler will be called whenever the user confirmed an image
    '  in the image selection control. We will need to update the business entities
    '  image property with the new value.
    '</summary>
    Private Sub WebImageSelectionPopUpButton1_ImageSelected(ByVal configuration
As Platform.WebUI.ImageConfiguration) Handles
WebImageSelectionPopUpButton1.ImageSelected

        If Me.BusinessEntity Is Nothing Then _
            Exit Sub
        Me.BusinessEntity.Image = configuration.ImageHtml
        txtImage.Text = configuration.ImageHtml

    End Sub 'WebImageSelectionPopUpButton1_ImageSelected
End Class 'ArticleAdminDetail
```

Appendix 10 – Article Administration User Control (HTML)

```
<%@ Control Language="vb" AutoEventWireup="false"
CodeBehind="ArticleAdmin.ascx.vb" Inherits="Primavera.KnowledgeBase.WebUI.ArticleAdmin"
%>
<!-- please not the corrected "Src" values for the "Register" tags; if you drag and
drop
     user controls to the designer page, please make sure that the "Src" attribute has
     the correct value - based on the final location on the production server. -->
<%@ Register TagPrefix="WebC" TagName="WebGridTableFilter"
Src="../Primavera.Platform/WebGridTableFilter.ascx" %>
<%@ Register TagPrefix="WebC" TagName="WebGridMessage"
Src="../Primavera.Platform/WebGridMessage.ascx" %>
<%@ Register Src="ArticleAdminDetail.ascx"
TagName="ArticleAdminDetail" TagPrefix="WebC" %>

<!-- the following user control will be used to issue messages to the user -->
<WebC:webgridmessage id="messageBox" runat="server"
visible="False"></WebC:webgridmessage>
<asp:Panel ID="panelArticleMaster" Visible="True" runat=server>
    <!-- this panel will handle the grid that shows all available business entities to
         the user. this "master" grid also implements common funcionalities like insert
         new item, edit/clone and remove the current item. this panel will be hidden by
         the code behind whenever the details (ArticleAdminDetail) needs to be shown. -
->
    <table id="tableArticles" runat="server" cellspacing="0" cellpadding="0"
width="100%" border="0">
        <tr>
            <td>
                <WebC:webgridtablefilter id="articleAdminMasterGrid" visible="True"
                    runat="server" width="100%" height="20px" DoPagination="True"
                     AllowDefaultPaging="True" AllowDefaultSorting="True"
                     AllowSelectingRecordsPerPage="True"
DefaultPagingMode="NumericPages"
                     DefaultPagingPosition="Top" PageSize="10" PageSizeIndex="0">
                </WebC:webgridtablefilter>
            </td>
        </tr>
    </table>
</asp:Panel>
<asp:Panel ID="panelArticleDetails" Visible="False" runat=server>
    <!-- this panel will handle the current entities details. it is shown whenever the
         user edits the currently selected or inserts a new record. we could have done
         this also in a separate aspx file. -->
    <table>
    <tr>
        <td>
            <WebC:ArticleAdminDetail id="articleAdminEntityDetails" runat="server">
```

```
            </WebC:ArticleAdminDetail></td>
        </td>
    </tr>
    </table>
</asp:Panel>
```

Appendix 11 – Article Administration User Control (VB)

```
Imports Primavera.Platform.WebUI
Imports Primavera.Platform.WebUI.Controls

Public Enum VisualizationMode
    Consultation
    Insertion
    Edition
End Enum

Partial Public Class ArticleAdmin
    Inherits Primavera.Platform.WebUI.WebComponentBase

    'declare user controls referenced from the PRIMAVERA WebCentral Platform
    'here in the code behind file, using the exact variable name as stated in the ascx
    Protected WithEvents messageBox As Primavera.Platform.WebUI.Controls.WebGridMessage
    Protected WithEvents articleAdminMasterGrid As
Primavera.Platform.WebUI.Controls.WebGridTableFilter

    '<summary>
    '  This UserControl must handle various modes, as it joins the master and
    '  detail form. The following property will persist the current state, using the
    '  Session object. The initial state is consultation, which means, the master
    '  grid view is displayed. Whenever the user edits or inserts a new record, the
    '  master grid needs to be hidden and the details pane needs to be shown.
    '</summary>
    Private Property Mode() As VisualizationMode
        Get
            If Session("ArticleAdminMode") Is Nothing Then
                Return VisualizationMode.Consultation
            Else
                Return CType(Session("ArticleAdminMode"), VisualizationMode)
            End If
        End Get
        Set(ByVal Value As VisualizationMode)
            Select Case Value
                Case VisualizationMode.Consultation
                    panelArticleMaster.Visible = True
                    panelArticleDetails.Visible = False
                Case VisualizationMode.Insertion
                    panelArticleMaster.Visible = False
                    panelArticleDetails.Visible = True
                    articleAdminEntityDetails.Mode = VisualizationMode.Insertion
                Case VisualizationMode.Edition
                    panelArticleMaster.Visible = False
                    panelArticleDetails.Visible = True
                    articleAdminEntityDetails.Mode = VisualizationMode.Edition
```

```
            End Select
            Session("ArticleAdminMode") = Value
        End Set
    End Property 'Mode

    '<summary>
    '  The following method is used to show a message (or an error) to the user;
    '  it will show the platform's message box user control and initialize the
    '  icon/text to be shown.
    '</summary>
    Private Sub ShowMessage(ByVal Type As Platform.WebUI.Controls.WebGridMessageType,
ByVal Title As String, ByVal Message As String)

        messageBox.Visible = True
        messageBox.MessageType = Type
        messageBox.MessageTitle = Title
        messageBox.MessageText = Message
        messageBox.Refresh()

    End Sub 'ShowMessage

    '<summary>
    '  The following method will be called when the page loads; it should be used
    '  to initialize the page's controls.
    '</summary>
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

        'initialize the table filter grid; we'll need to initialize this control with
module name
        'as well as table filter object (defined in the solutions Modules project),
which will be
        'used to query the information from the database.
        articleAdminMasterGrid.ModuleName = "Primavera.KnowledgeBase"
        articleAdminMasterGrid.TableFilterID = New Guid("{25662350-E994-45dd-9C75-
340AA517357D}")

        'initialize the details user control's web context property
        articleAdminEntityDetails.WebContext = Me.WebContext

        'if not postback: initialize the forms mode to its default (consultation)
        If Not Me.IsPostBack Then
            Mode = VisualizationMode.Consultation
            articleAdminMasterGrid.RefreshDataSource()
        End If
    End Sub 'Page_Load

    '<summary>
    '  The following method will be called whenever the user clicked on the "New"
button
```

```
    '  of the table filter grid. The method will simply change the mode to insertion,
which
    '  will cause the table filter grid to be hidden and the details view to be shown.
    '</summary>
    Private Sub articleAdminMasterGrid_CreateRecord(ByVal Sender As Object)
Handles articleAdminMasterGrid.CreateRecord

        Me.Mode = VisualizationMode.Insertion

    End Sub 'articleAdminMasterGrid_CreateRecord

    '<summary>
    '  The following method will be called whenever the user clicked on the "Edit"
button
    '  of the table filter grid. The method will initialize the details view unique id
    '  property and set the edition mode, which will cause the table filter to be
hidden
    '  and the details view to be shown.
    '</summary>
    Private Sub articleAdminMasterGrid_EditRecord(ByVal Sender As Object,
ByVal selectedRowIdentifier As
Platform.WebUI.Controls.WebGridTableFilter.TableFilterKeyValueCollection,
ByRef cancel As Boolean) Handles articleAdminMasterGrid.EditRecord

        Try
            articleAdminEntityDetails.ArticleId = selectedRowIdentifier("ID")
            Mode = VisualizationMode.Edition

        Catch ex As System.Security.SecurityException
            Mode = VisualizationMode.Consultation
            ShowMessage(WebGridMessageType.Warning, String.Empty, ex.Message)

        Catch ex As Exception
            Mode = VisualizationMode.Consultation
            ShowMessage(WebGridMessageType.Error, String.Empty, ex.Message)

        End Try

    End Sub 'articleAdminMasterGrid_EditRecord

    '<summary>
    '  The following method will be called whenever the user clicked on the "Clone"
button
    '  of the table filter grid. The code here demonstrates how to clone a existing
record.
    '</summary>
    Private Sub articleAdminMasterGrid_CloneRecord(ByVal Sender As Object,
ByVal selectedRowIdentifier As
Platform.WebUI.Controls.WebGridTableFilter.TableFilterKeyValueCollection,
ByRef cancel As Boolean) Handles articleAdminMasterGrid.CloneRecord
```

```
        Try
            Dim articleId As System.Guid = CType(selectedRowIdentifier("ID"),
System.Guid)
            Dim businessLayer As New Business.Articles
            businessLayer.Clone(WebContext.User.EngineContext, articleId)
            articleAdminMasterGrid.RefreshDataSource()

        Catch ex As System.Security.SecurityException
            ShowMessage(WebGridMessageType.Warning, String.Empty, ex.Message)

        Catch ex As Exception
            ShowMessage(WebGridMessageType.Error, String.Empty, ex.Message)

        End Try

    End Sub 'articleAdminMasterGrid_CloneRecord

    '<summary>
    '  The following method will be called whenever the user clicked on the "Delete"
button,
    '  and confirmed the client-side confirmation message. The code here shows how to
delete
    '  the currently selected record.
    '</summary>
    Private Sub articleAdminMasterGrid_DeleteRecord(ByVal Sender As Object,
ByVal selectedRowIdentifier As
Platform.WebUI.Controls.WebGridTableFilter.TableFilterKeyValueCollection,
ByRef cancel As Boolean) Handles articleAdminMasterGrid.DeleteRecord

        Try
            Dim articleId As System.Guid = CType(selectedRowIdentifier("ID"),
System.Guid)
            Dim businessLayer As New Business.Articles
            businessLayer.Remove(WebContext.User.EngineContext, articleId)
            articleAdminMasterGrid.RefreshDataSource()

        Catch ex As System.Security.SecurityException
            ShowMessage(WebGridMessageType.Warning, String.Empty, ex.Message)

        Catch ex As Exception
            ShowMessage(WebGridMessageType.Error, String.Empty, ex.Message)

        End Try

    End Sub 'articleAdminMasterGrid_DeleteRecord

    '<summary>
    '  The following method will be called whenever the user clicked on the "Refresh"
button.
```

```
    '  It will simply refresh the master grid and bind it to the data source.
    '</summary>
    Private Sub articleAdminMasterGrid_Refresh(ByRef Cancel As Boolean)
Handles articleAdminMasterGrid.Refresh

        articleAdminMasterGrid.RefreshDataSource()

    End Sub 'articleAdminMasterGrid_Refresh

    '<summary>
    '  The following method will be called whenever the user clicked on the "Confirm"
button
    '  in the details view. This method sets the mode to consultation, which will cause
the
    '  table filter grid to be shown and the details control to be hidden.
    '</summary>
    Private Sub articleAdminEntityDetails_ArticleSaved()
Handles articleAdminEntityDetails.ArticleSaved

        Mode = VisualizationMode.Consultation
        articleAdminMasterGrid.RefreshDataSource()

    End Sub 'articleAdminEntityDetails_ArticleSaved

    '<summary>
    '  The following method will be called whenever the user clicked on the "Back"
button in
    '  the details view. This method simply sets the mode to consultation, which will
cause the
    '  table filter grid to be shown and the details control to be hidden.
    '</summary>
    Private Sub articleAdminEntityDetails_BackButtonClicked()
Handles articleAdminEntityDetails.BackButtonClicked

        Mode = VisualizationMode.Consultation

    End Sub 'articleAdminEntityDetails_BackButtonClicked
End Class 'ArticleAdmin.ascx
```

Appendix 12 – PropertyBag Entity Class (ArticleList.vb)

```
Imports Primavera.Platform.Engine.Entities

Namespace PropertyBags
    <Serializable(), SerializationEntity("{A461C193-98FB-4a52-94AF-7EB6574B470E}",
"1.0")> _
    Public Class ArticleList
        Inherits PropertyBagBase

        Private mNumArticles As Integer
        Private mOrder As ArticleListOrder
        Private mIncludeImage As Boolean
        Private mIncludeSummary As Boolean
        Private mIncludeAuthor As Boolean
        Private mDetailPageID As Guid

        Public Property NumArticles() As Integer
            Get
                Return mNumArticles
            End Get
            Set(ByVal Value As Integer)
                mNumArticles = Value
            End Set
        End Property

        Public Property IncludeImage() As Boolean
            Get
                Return mIncludeImage
            End Get
            Set(ByVal Value As Boolean)
                mIncludeImage = Value
            End Set
        End Property

        Public Property IncludeSummary() As Boolean
            Get
                Return mIncludeSummary
            End Get
            Set(ByVal Value As Boolean)
                mIncludeSummary = Value
            End Set
        End Property

        Public Property IncludeAuthor() As Boolean
            Get
                Return mIncludeAuthor
```

```
            End Get
            Set(ByVal Value As Boolean)
                mIncludeAuthor = Value
            End Set
        End Property

        Public Property Order() As ArticleListOrder
            Get
                Return mOrder
            End Get
            Set(ByVal Value As ArticleListOrder)
                mOrder = Value
            End Set
        End Property

        Public Property DetailPageID() As Guid
            Get
                Return mDetailPageID
            End Get
            Set(ByVal Value As Guid)
                mDetailPageID = Value
            End Set
        End Property

    End Class

    Public Enum ArticleListOrder
        OrderByDate
        OrderByAuthor
        OrderByTitle
    End Enum

End Namespace
```

Appendix 13 – Remoting Layer ArticleList.vb

```
Imports Primavera.Platform.Engine.Entities
Imports Primavera.Platform.Engine.IRemoting

Namespace PropertyBags
    Public Class ArticleList
        Inherits Primavera.Platform.Engine.Remoting.RemoteServiceBase
        Implements IRemoting.PropertyBags.IArticleList

        Public Function Edit(ByVal RemoteContext As RemoteEngineContext, _
ByVal ID As System.Guid) As Entities.PropertyBags.ArticleList _
             Implements IRemoting.PropertyBags.IArticleList.Edit


            Dim BSO As New Business.PropertyBags.ArticleList
            Return BSO.Edit(RemoteContext.EngineContext, ID)
        End Function

        Public Sub Update(ByVal RemoteContext As RemoteEngineContext, _
ByVal PropertyBag As Entities.PropertyBags.ArticleList) _
Implements IRemoting.PropertyBags.IArticleList.Update

            Dim BSO As New Business.PropertyBags.ArticleList
            BSO.Update(RemoteContext.EngineContext, PropertyBag)
        End Sub

        Public Function Exists(ByVal RemoteContext As RemoteEngineContext, _
ByVal ID As System.Guid) As System.Boolean _
Implements IRemoting.PropertyBags.IArticleList.Exists

            Dim BSO As New Business.PropertyBags.ArticleList
            Return BSO.Exists(RemoteContext.EngineContext, ID)
        End Function

        Public Sub Remove(ByVal RemoteContext As RemoteEngineContext, _
ByVal ID As System.Guid) _
Implements IRemoting.PropertyBags.IArticleList.Remove

            Dim BSO As New Business.PropertyBags.ArticleList
            BSO.Remove(RemoteContext.EngineContext, ID)
        End Sub

        Public Function Clone(ByVal RemoteContext As RemoteEngineContext, _
ByVal ID As System.Guid) As System.Guid _
Implements IRemoting.PropertyBags.IArticleList.Clone

            Dim BSO As New Business.PropertyBags.ArticleList
            Return BSO.Clone(RemoteContext.EngineContext, ID)
```

```
        End Function
    End Class
End Namespace
```

Appendix 14 – The Configuration Dialog (ArticleList.vb)

```
Imports Primavera.Platform.Engine.IRemoting
Imports Primavera.KnowledgeBase.Entities
Imports Primavera.KnowledgeBase.IRemoting

Public Class ArticleList
    '<summary>
    '  The following private attribute will store the article lists current
    '  configuration settings.
    '</summary>
    Private articleListPropertyBag As Entities.PropertyBags.ArticleList
    Private componentLink As Platform.WinUI.ComponentLinkDialogOutput

    '<summary>
    '  custom c'tor
    '</summary>
    Public Sub New(ByVal WinContext As WinContext)
        MyBase.New(WinContext)
        InitializeComponent()
    End Sub 'New

    '<summary>
    '  Call the following method to configure the properties for a new
    '  component.
    '</summary>
    Public Function INew() As Guid
        InitializeForm()
        NewPropertyBag()
        ShowPropertyBag()
        If Me.ShowDialog() = DialogResult.OK Then
            Return articleListPropertyBag.ID
        Else
            Return Nothing
        End If
    End Function 'INew

    '<summary>
    '  Call the following method to configure the properties for an already
    '  existing component.
    '</summary>
    Public Sub IEdit(ByVal PropertyBagID As Guid)
        InitializeForm()
        EditPropertyBag(PropertyBagID)
        ShowPropertyBag()
        Me.ShowDialog()
    End Sub 'IEdit

    '<summary>
```

```
    '  The following method is called to initialize a new property bag
    '  entity, and is called whenever the user places a new component
    '  on a page.
    '</summary>
    Private Sub NewPropertyBag()
        articleListPropertyBag = New Entities.PropertyBags.ArticleList
        With articleListPropertyBag
            .ID = Guid.NewGuid
            .IncludeAuthor = True
            .IncludeImage = True
            .IncludeSummary = True
            .NumArticles = 5
            .Order = Entities.PropertyBags.ArticleListOrder.OrderByDate
            .DetailPageID = Guid.Empty
        End With
    End Sub

    '<summary>
    '  The following method will be called to read an existing property
    '  bag from the WebCentral database.
    '</summary>
    Private Sub EditPropertyBag(ByVal PropertyBagID As Guid)
        Dim Context As Platform.Engine.IRemoting.RemoteEngineContext =
MyBase.CreateRemoteEngineContext()
        Dim Proxy As IRemoting.IProxy = Me.CreateRemoteProxy(GetType(IRemoting.IProxy))
        articleListPropertyBag = Proxy.PropertyBags.ArticleList.Edit(Context,
PropertyBagID)
    End Sub

    '<summary>
    '  The following method will be called to store the property bags
    '  attributes to the WebCentral database.
    '</summary>
    Private Sub UpdatePropertyBag()
        Dim Context As Platform.Engine.IRemoting.RemoteEngineContext =
MyBase.CreateRemoteEngineContext()
        Dim Proxy As IRemoting.IProxy = Me.CreateRemoteProxy(GetType(IRemoting.IProxy))
        Proxy.PropertyBags.ArticleList.Update(Context, articleListPropertyBag)
    End Sub

    '<summary>
    '  The following method will be called to initialize the dialog's
    '  user interface controls with the property bag's values.
    '</summary>
    Private Sub ShowPropertyBag()
        With articleListPropertyBag
            txtArticleCount.Value = .NumArticles
            cmbArticleSort.SelectedIndex = .Order
            chkShowAuthor.Checked = .IncludeAuthor
            chkShowImage.Checked = .IncludeImage
```

```
            chkShowSummary.Checked = .IncludeSummary
            componentLink = Platform.WinUI.Dialogs.ComponentLinkEdit(WinContext,
.DetailPageID)
            If Not componentLink Is Nothing Then
                txtArticleDetailLink.Text = componentLink.PageName
            Else
                txtArticleDetailLink.Text = "<Default>"
            End If
        End With
    End Sub

    '<summary>
    '  The following method will be called to read the values in the
    '  dialog's user interface to to property bag object.
    '</summary>
    Private Sub ReadPropertyBag()
        With articleListPropertyBag
            .NumArticles = txtArticleCount.Value
            .Order = cmbArticleSort.SelectedIndex
            .IncludeAuthor = chkShowAuthor.Checked
            .IncludeImage = chkShowImage.Checked
            .IncludeSummary = chkShowSummary.Checked
            .DetailPageID = componentLink.PageID
        End With
    End Sub

    '<summary>
    '  The following method initializes the dialogs controls
    '</summary>
    Private Sub InitializeForm()
        cmbArticleSort.Items.Clear()
        cmbArticleSort.Items.Add("Article Date")
        cmbArticleSort.Items.Add("Article Author")
        cmbArticleSort.Items.Add("Article Title")
        cmbArticleSort.SelectedIndex = 0

        txtArticleCount.Maximum = 15
        txtArticleCount.Minimum = 1
    End Sub

    '<summary>
    '  The following method is called whenever the user clicked the Confirm
    '  button. It will shift the configuration data from the user interface
    '  to the property bag object and store the changes in the Enterprise
    '  Portals database.
    '</summary>
    Private Sub btnConfirm_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdConfirm.Click
        ReadPropertyBag()
        UpdatePropertyBag()
```

```
        Me.DialogResult = DialogResult.OK
    End Sub

    '<summary>
    '  The following method is called whenever the user clicked the Cancel
    '  button. The user interface closes and no changes are done.
    '</summary>
    Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdCancel.Click
        Me.DialogResult = DialogResult.Cancel
    End Sub

    '<summary>
    '  The following method is called whenever the user clicked the Browse
    '  button, to select a page that contains the ArticleDetail component.
    '</summary>
    Private Sub btnLinkDetalhe_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdBrowseHyperlink.Click
        Dim dlgInput As New Platform.WinUI.ComponentLinkDialogInput()
        dlgInput.ComponentID = New Guid("{B6B669F2-35B2-43a1-A03D-7325EEB2B082}")
        dlgInput.PortalID = WinContext.Portal.PortalID
        dlgInput.PageID = articleListPropertyBag.DetailPageID
        Dim dlgOutput As Platform.WinUI.ComponentLinkDialogOutput =
Primavera.Platform.WinUI.Dialogs.ComponentLinkSelect(WinContext, dlgInput)
        If Not dlgOutput Is Nothing Then
            articleListPropertyBag.DetailPageID = dlgOutput.PageID
            ShowPropertyBag()
        End If
    End Sub
End Class
```

Appendix 15 – Culture Dependent Entity (ArticleCulture.vb)

```
Imports Primavera.Platform.Engine.Entities

<Serializable(), BusinessEntityAttribute("KNB_ArticleCultures","4D741C19-2237-4691-
997F-F943799438D3")> _
Public Class ArticleCulture
    Inherits BusinessEntityBase

    Private mArticleID As Guid
    Private mCultureID As Guid
    Private mTitle As String
    Private mSummary As String
    Private mBody As String

    <BusinessEntityField("ArticleID")> _
    Public Property ArticleID() As Guid
        Get
            Return mArticleID
        End Get
        Set(ByVal Value As Guid)
            mArticleID = Value
        End Set
    End Property

    <BusinessEntityField("CultureID")> _
    Public Property CultureID() As Guid
        Get
            Return mCultureID
        End Get
        Set(ByVal Value As Guid)
            mCultureID = Value
        End Set
    End Property

    <BusinessEntityField("Title")> _
    Public Property Title() As String
        Get
            Return mTitle
        End Get
        Set(ByVal Value As String)
            mTitle = Value
        End Set
    End Property

    <BusinessEntityField("Summary")> _
    Public Property Summary() As String
        Get
```

```
            Return mSummary
        End Get
        Set(ByVal Value As String)
            mSummary = Value
        End Set
    End Property

    <BusinessEntityField("Body")> _
    Public Property Body() As String
        Get
            Return mBody
        End Get
        Set(ByVal Value As String)
            mBody = Value
        End Set
    End Property

End Class
```

Appendix 16 – Culture Dependent Entity (ArticleCultures.vb)

```
Imports Primavera.Platform.Engine.Entities

<Serializable()> _
Public Class ArticleCultures
    Inherits CollectionEntityBase

    Public Sub Add(ByVal Value As ArticleCulture)
        Me.List.Add(Value)
    End Sub

    Default Public Property Item(ByVal Index As Integer) As ArticleCulture
        Get
            Return CType(Me.List.Item(Index), ArticleCulture)
        End Get
        Set(ByVal Value As ArticleCulture)
            Me.List.Item(Index) = Value
        End Set
    End Property

    Default Public Property Item(ByVal ID As Guid) As ArticleCulture
        Get
            Dim Entity As ArticleCulture
            For Each Entity In Me.List
                If Entity.ID.Equals(ID) Then Return Entity
            Next
            Return Nothing
        End Get
        Set(ByVal Value As ArticleCulture)
            Dim Entity As ArticleCulture
            For Each Entity In Me.List
                If Entity.ID.Equals(ID) Then
                    Entity = Value
                    Exit Property
                End If
            Next
            Add(Value)
        End Set
    End Property

    Public ReadOnly Property GetItemByArticleID(ByVal ArticleID As Guid) As
ArticleCulture
        Get
            Dim Entity As ArticleCulture
            For Each Entity In Me.List
                If Entity.ArticleID.Equals(ArticleID) Then Return Entity
            Next
```

```
            Return Nothing
        End Get
    End Property

    Public ReadOnly Property GetItemByCultureID(ByVal CultureID As Guid) As
ArticleCulture
        Get
            Dim Entity As ArticleCulture
            For Each Entity In Me.List
                If Entity.CultureID.Equals(CultureID) Then Return Entity
            Next
            Return Nothing
        End Get
    End Property
End Class
```

Appendix 17 – SQL Database Update Script

```
if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[FK_KNB_ArticleCultures_KNB_Articles]') and OBJECTPROPERTY(id,
N'IsForeignKey') = 1)
ALTER TABLE [dbo].[KNB_ArticleCultures] DROP CONSTRAINT
FK_KNB_ArticleCultures_KNB_Articles
GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[KNB_ArticleCultures]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[KNB_ArticleCultures]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[KNB_Articles]')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[KNB_Articles]
GO

CREATE TABLE [dbo].[KNB_ArticleCultures] (
     [ID] [uniqueidentifier] NOT NULL ,
     [ArticleID] [uniqueidentifier] NULL ,
     [CultureID] [uniqueidentifier] NULL ,
     [Title] [nvarchar] (250)  NULL ,
     [Summary] [nvarchar] (500) NULL ,
     [Body] [nvarchar] (500) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[KNB_Articles] (
     [ID] [uniqueidentifier] NOT NULL ,
     [Date] [datetime] NULL ,
     [Author] [nvarchar] (100) NULL ,
     [Image] [nvarchar] (250)  NULL ,
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[KNB_ArticleCultures] WITH NOCHECK ADD
     CONSTRAINT [PK_KNB_ArticleCultures] PRIMARY KEY  CLUSTERED
     (
          [ID]
     )  ON [PRIMARY]
GO

ALTER TABLE [dbo].[KNB_Articles] WITH NOCHECK ADD
     CONSTRAINT [PK_KNB_Articles] PRIMARY KEY  CLUSTERED
     (
```

```
          [ID]
     )  ON [PRIMARY]
GO

ALTER TABLE [dbo].[KNB_ArticleCultures] ADD
     CONSTRAINT [FK_KNB_ArticleCultures_KNB_Articles] FOREIGN KEY
     (
          [ArticleID]
     ) REFERENCES [dbo].[KNB_Articles] (
          [ID]
     ),
     CONSTRAINT [FK_KNB_ArticleCultures_PLT_Cultures] FOREIGN KEY
     (
          [CultureID]
     ) REFERENCES [dbo].[PLT_Cultures] (
          [ID]
     )
GO

DELETE FROM KNB_ArticleCultures
DELETE FROM KNB_Articles

DECLARE @ArticleID uniqueidentifier
SET @ArticleID = NEWID()
INSERT INTO KNB_Articles VALUES (@ArticleID,GETDATE(),'Jim Button','<img
src="UserFiles/Images/003.bmp" >')
INSERT INTO KNB_ArticleCultures VALUES (NEWID(),@ArticleID,'EA16D9A6-D662-4FEC-AB32-
A2EA62D6F2CB','Desenvolvendo um Módulo','Este artigo descreve...','Para desenvolver um
Módulo ePrimavera é imprescindivel começar por uma leitura adequada...')
INSERT INTO KNB_ArticleCultures VALUES (NEWID(),@ArticleID,'9F9C7468-22A3-4414-B8F5-
596346EF4300','Developing an Module','This article describes ...','To build an new
ePrimavera Module, you should begin with a proper reading into ...')
SET @ArticleID = NEWID()
INSERT INTO KNB_Articles VALUES (@ArticleID,GETDATE(),'Laura Jones','<img
src="UserFiles/Images/002.bmp" >')
INSERT INTO KNB_ArticleCultures  VALUES (NEWID(),@ArticleID,'EA16D9A6-D662-4FEC-AB32-
A2EA62D6F2CB','Integração','Discutindo a integração  ...','Este assunto é susceptivel
de levantar alguma polemica. Apesar de existir bastante documentação sobre este tema
...')
INSERT INTO KNB_ArticleCultures VALUES (NEWID(),@ArticleID,'9F9C7468-22A3-4414-B8F5-
596346EF4300','How to integrate','Talking about .NET integration ...','This subject its
prone to some hot discussion, although there''s enough documentation on this theme
...')
```

Appendix 18 – Approval Workflow (ApprovalInstances.vb)

```
Imports System.Resources
Imports Primavera.Platform.Engine.Entities
Imports Primavera.Platform.Modules.Entities
Imports Primavera.Platform.Services.Entities

Public Class ApprovalInstances
    Implements IApprovalInstances

    Private MyModuleID As New Guid("{39af2e6e-e68e-4111-9410-24de9682421d}")

    Public Function GetApprovalsForUser(ByVal objContext As EngineContext) As
ApprovalsPending _
        Implements IApprovalInstances.GetApprovalsForUser
        Dim objApprovalServices As New
Primavera.Platform.Services.Business.ApprovalServices
        Dim objApprovalsPending As ApprovalsPending
        Dim objApprovalPending As ApprovalPending

        objApprovalsPending = objApprovalServices.GetApprovalInstancesPendingForUser( _
objContext, MyModuleID)
        If (Not objApprovalsPending Is Nothing) Then
            For Each objApprovalPending In objApprovalsPending
                With objApprovalPending
                    Dim guidID As Guid
                    Dim strTitle As String = String.Empty
                    GetContentIDTitle(objContext, .ApprovalID, .TypeID, guidID,
strTitle)
                    .Title = strTitle
                    .TypeName = GetApprovalTypeName(.TypeID)
                    .URL = GetApprovalURL(.TypeID, guidID)
                End With
            Next
        End If

        Return objApprovalsPending

    End Function

    Public Function CanPerformApproval(ByVal objContext As EngineContext, _
        ByVal objApprovalInstance As ApprovalInstance, _
        ByVal objCurrentState As ApprovalRuleState, _
        ByVal objCurrentApproval As ApprovalRuleApproval, _
        ByVal objApprovedState As ApprovalRuleState, _
        ByRef strErrors As String) As Boolean _
        Implements IApprovalInstances.CanPerformApproval
```

```
        Return True

    End Function

    Public Function CanPerformRejection(ByVal objContext As EngineContext, _
        ByVal objApprovalInstance As ApprovalInstance, _
        ByVal objCurrentState As ApprovalRuleState, _
        ByVal objCurrentApproval As ApprovalRuleApproval, _
        ByVal objApprovedState As ApprovalRuleState, _
        ByRef strErrors As String) As Boolean _
        Implements IApprovalInstances.CanPerformRejection

        Return True

    End Function

    Public Function PerformApproval(ByVal objContext As EngineContext, _
        ByVal objApprovalInstance As ApprovalInstance, _
        ByVal objCurrentState As ApprovalRuleState, _
        ByVal objCurrentApproval As ApprovalRuleApproval, _
        ByVal objApprovedState As ApprovalRuleState, _
        ByRef strErrors As String) As Boolean _
        Implements IApprovalInstances.PerformApproval

        'Unlock and publish message (only at the end)

        If (objApprovedState.StateType = ApprovalRuleStateTypeEnum.arstEndApproved)
Then

            objContext.BeginTransaction()
            Try
                UnlockAndPublish(objContext, objApprovalInstance.ID, _
objApprovalInstance.ApprovalTypeID, True)
                objContext.CommitTransaction()
            Catch ex As Exception
                strErrors = ex.Message
                objContext.RollBackTransaction()
                Return False
            End Try

        End If

        'OK

        Return True

    End Function

    Public Function PerformRejection(ByVal objContext As EngineContext, _
        ByVal objApprovalInstance As ApprovalInstance, _
```

```
        ByVal objCurrentState As ApprovalRuleState, _
        ByVal objCurrentApproval As ApprovalRuleApproval, _
        ByVal objRejectedState As ApprovalRuleState, _
        ByRef strErrors As String) As Boolean _
        Implements IApprovalInstances.PerformRejection

        'Unlock and publish message (only at the end)

        If (objRejectedState.StateType = ApprovalRuleStateTypeEnum.arstEndRejected)
Then

            objContext.BeginTransaction()
            Try
                UnlockAndPublish(objContext, objApprovalInstance.ID, _
objApprovalInstance.ApprovalTypeID, False)
                objContext.CommitTransaction()
            Catch ex As Exception
                strErrors = ex.Message
                objContext.RollBackTransaction()
                Return False
            End Try

        End If

        'OK

        Return True

    End Function

    Public Function GetContentInfo(ByVal objContext As EngineContext, _
        ByVal guidTypeID As System.Guid, _
        ByVal guidApprovalID As System.Guid) As ApprovalContentInfo _
        Implements IApprovalInstances.GetContentInfo

        Dim guidID As Guid
        Dim strTitle As String = String.Empty
        Dim strURL As String = String.Empty

        GetContentIDTitle(objContext, guidApprovalID, guidTypeID, guidID, strTitle)
        strURL = GetApprovalURL(guidTypeID, guidID)

        Dim objRes As New ApprovalContentInfo
        With objRes
            .ModuleID = MyModuleID
            .TypeID = guidTypeID
            .ID = guidID
            .Title = strTitle
            .URL = strURL
        End With
```

```
        Return objRes

    End Function

    Public Function TranslateSpecialGroup(ByVal objContext As EngineContext, _
        ByVal guidTypeID As System.Guid, _
        ByVal guidSpecialGroupID As System.Guid, _
        ByVal guidUserID As System.Guid, _
        ByVal strTagInfo As String) As System.Guid _
        Implements IApprovalInstances.TranslateSpecialGroup

        Return Nothing

    End Function

    Private Sub UnlockAndPublish(ByVal objContext As EngineContext, _
        ByVal guidApprovalID As Guid, _
        ByVal guidTypeID As Guid, _
        ByVal blnPublish As Boolean)

        'Get content ID

        Dim guidID As Guid
        Dim strTitle As String = String.Empty
        GetContentIDTitle(objContext, guidApprovalID, guidTypeID, guidID, strTitle)

        Dim objProxy As New Primavera.KnowledgeBase.Business.Proxy

        If guidTypeID.Equals(ApprovalTypesIDs.Articles) Then

            Dim objBSO As New Business.Articles
            objBSO.UnLock(objContext, guidID)
            If blnPublish Then objBSO.Publish(objContext, guidID)
        End If

    End Sub

    Private Sub GetContentIDTitle(ByVal objContext As EngineContext, _
        ByVal guidApprovalID As Guid, _
        ByVal guidTypeID As Guid, _
        ByRef guidContentID As Guid, _
        ByRef strTitle As String)

        If guidTypeID.Equals(ApprovalTypesIDs.Articles) Then
            Dim objBSO As New Business.Articles
            objBSO.GetContentFromRuleID(objContext, guidApprovalID, guidContentID,
strTitle)
        End If
```

```
    End Sub

    Private Function GetApprovalTypeName(ByVal guidTypeID As Guid) As String

        If guidTypeID.Equals(ApprovalTypesIDs.Articles) Then
            Return "Categorias de Artigos"
        Else
            Return ""
        End If

    End Function

    Private Function GetApprovalURL(ByVal guidTypeID As Guid, _
        ByVal guidContentID As Guid) As String

        If guidTypeID.Equals(ApprovalTypesIDs.Articles) Then
            Return "ComponentID=" + ComponentsIDs.KnowledgeBaseArticle.ToString _
                + "&ArticleID=" + guidContentID.ToString()
        Else
            Return String.Empty
        End If

    End Function
End Class
```

Appendix 19 – SQL Database Update Script

```
    if exists (select * from dbo.sysobjects
where id = object_id(N'[dbo].[FK_KNB_ArticleCultures_KNB_Articles]')
and OBJECTPROPERTY(id, N'IsForeignKey') = 1)
    ALTER TABLE [dbo].[KNB_ArticleCultures] DROP CONSTRAINT
FK_KNB_ArticleCultures_KNB_Articles
    GO

    if exists (select * from dbo.sysobjects
where id = object_id(N'[dbo].[KNB_ArticleCultures]')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
    drop table [dbo].[KNB_ArticleCultures]
    GO

    if exists (select * from dbo.sysobjects
where id = object_id(N'[dbo].[KNB_Articles]')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
    drop table [dbo].[KNB_Articles]
    GO

    CREATE TABLE [dbo].[KNB_ArticleCultures] (
        [ID] [uniqueidentifier] NOT NULL ,
        [ArticleID] [uniqueidentifier] NULL ,
        [CultureID] [uniqueidentifier] NULL ,
        [Title] [nvarchar] (250)  NULL ,
        [Summary] [nvarchar] (500) NULL ,
        [Body] [nvarchar] (500) NULL
    ) ON [PRIMARY]
    GO

    CREATE TABLE [dbo].[KNB_Articles] (
        [ID] [uniqueidentifier] NOT NULL ,
        [Article] [nvarchar] (50) NOT NULL ,
        [Date] [datetime] NULL ,
        [Author] [nvarchar] (100) NULL ,
        [Image] [nvarchar] (250)  NULL ,
        [Published] [bit] NULL ,
        [Locked] [bit] NULL ,
        [ApprovalID] [uniqueidentifier] NULL
    ) ON [PRIMARY]
    GO

    ALTER TABLE [dbo].[KNB_ArticleCultures] WITH NOCHECK ADD
        CONSTRAINT [PK_KNB_ArticleCultures] PRIMARY KEY  CLUSTERED
        (
            [ID]
        )  ON [PRIMARY]
```

```
    GO

    ALTER TABLE [dbo].[KNB_Articles] WITH NOCHECK ADD
        CONSTRAINT [PK_KNB_Articles] PRIMARY KEY  CLUSTERED
        (
            [ID]
        )  ON [PRIMARY]
    GO

    ALTER TABLE [dbo].[KNB_ArticleCultures] ADD
        CONSTRAINT [FK_KNB_ArticleCultures_KNB_Articles] FOREIGN KEY
        (
            [ArticleID]
        ) REFERENCES [dbo].[KNB_Articles] (
            [ID]
        ),
        CONSTRAINT [FK_KNB_ArticleCultures_PLT_Cultures] FOREIGN KEY
        (
            [CultureID]
        ) REFERENCES [dbo].[PLT_Cultures] (
            [ID]
        )
    GO

    DELETE FROM KNB_ArticleCultures
    DELETE FROM KNB_Articles

    DECLARE @ArticleID1 uniqueidentifier
    SET @ArticleID1 = NEWID()
    INSERT INTO KNB_Articles VALUES (@ArticleID1,'Article 1',GETDATE(),'Jim Button',
'<img src="UserFiles/Images/003.bmp" >',1,0,NULL)
    INSERT INTO KNB_ArticleCultures VALUES (NEWID(),@ArticleID1,
'EA16D9A6-D662-4FEC-AB32-A2EA62D6F2CB','Desenvolvendo um Módulo',
'Este artigo descreve...',
'Para desenvolver um Módulo ePrimavera é imprescindivel começar por uma leitura
adequada...')
    INSERT INTO KNB_ArticleCultures VALUES (NEWID(),@ArticleID1,
'9F9C7468-22A3-4414-B8F5-596346EF4300','Developing an Module',
'This article describes ...','
To build an new ePrimavera Module, you should begin with a proper reading into ...')

    DECLARE @ArticleID2 uniqueidentifier
    SET @ArticleID2 = NEWID()
    INSERT INTO KNB_Articles VALUES (@ArticleID2,'Article 2',GETDATE(),'Laura Jones',
'<img src="UserFiles/Images/002.bmp" >',1,0,NULL)
    INSERT INTO KNB_ArticleCultures  VALUES (NEWID(),@ArticleID2,
'EA16D9A6-D662-4FEC-AB32-A2EA62D6F2CB','Integração',
'Discutindo a integração  ...',
'Este assunto é susceptivel de levantar alguma polemica. Apesar de existir bastante
documentação sobre este tema ...')
```

```
    INSERT INTO KNB_ArticleCultures VALUES (NEWID(),@ArticleID2,
'9F9C7468-22A3-4414-B8F5-596346EF4300','How to integrate',
'Talking about .NET integration ...',
'This subject its prone to some hot discussion, although there''s enough documentation
on this theme ...')

    DECLARE @CategoryTypeID uniqueidentifier
    SET @CategoryTypeID = 'B9F475F4-2C54-463f-8FD9-F6318EDAE971'

    DELETE FROM PLT_CategoryCultures
WHERE CategoryID IN (SELECT ID FROM PLT_Categories WHERE CategoryTypeID =
@CategoryTypeID)
    DELETE FROM PLT_CategoryPermissions
WHERE CategoryID IN (SELECT ID FROM PLT_Categories WHERE CategoryTypeID =
@CategoryTypeID)
    DELETE FROM PLT_Categories
WHERE ID IN (SELECT ID FROM PLT_Categories WHERE CategoryTypeID = @CategoryTypeID)

    DECLARE @CategoryID uniqueidentifier
    SET @CategoryID = NEWID()
    INSERT INTO PLT_Categories
([ID],CategoryTypeID,Category,ParentID,FullName,ApprovalRuleID)
VALUES (@CategoryID,@CategoryTypeID,'Artigos técnicos',NULL,'\Artigos técnicos\',NULL)
    INSERT INTO PLT_CategoryCultures ([ID],CategoryID,CultureID,[Name])
VALUES (NEWID(),@CategoryID,'EA16D9A6-D662-4FEC-AB32-A2EA62D6F2CB','Artigos técnicos')
    INSERT INTO PLT_CategoryPermissions
VALUES (NEWID(),@CategoryID ,1,'E8220F2B-01A9-4A04-A4A9-21DE20CD3325',1,1,1,1,1)
    INSERT INTO PLT_CategoryPermissions
VALUES (NEWID(),@CategoryID ,1,'4AA6432C-40A8-457A-9555-5021814CB96C',1,1,1,1,1)
    INSERT INTO PLT_CategoryPermissions
VALUES (NEWID(),@CategoryID ,1,'866B5E19-E098-4838-B51C-86BC522E9FD1',1,1,1,1,1)

    INSERT INTO PLT_EntityCategories VALUES (NEWID(),@ArticleID1,@CategoryID)
    INSERT INTO PLT_EntityCategories VALUES (NEWID(),@ArticleID2,@CategoryID)
```