Technical Note

Subclassing and Customizing REP++ Objects

Author: R&D Department

Publication date: December 16, 2006

Revised date: May 2010

Subclassing and Customizing REP++ Objects

Overview

REP++ provides a library of objects that can be used to represent different types of data. In order to be able to customize these objects or to create your own, REP++ allows their subclassing.

This document explains how to subclass and customize REP++ objects with the **Field** object as example.

Note: code samples are given in Visual Basic®.

Subclassing REP++ objects

REP++ provides a mechanism to subclass the different objects in its hierarchy. Subclassing allows the user to customize these objects by adding methods and properties or by overriding the basic methods to change the default behaviour of the object.

This subclassing mechanism is implemented using the class factory design pattern. The class factory allows the creation of the different custom objects. To use this mechanism, you should:

- Define your REP++ subclass object.
- Define a class factory.
- Register this class factory in the REP++ **Application** object. Once registered, the class factory becomes responsible for creating the REP++ objects, and the REP++ **Application** will delegate the task of creating new instances to this class.

Defining a REP++ subclass

A custom REP++ object should inherit from a REP++ object and define a constructor with an integer parameter. This parameter represents the kernel object. For example, you can subclass the **RepPP.FieldDef** class in order to add a new property. This property may return the value of a user-defined attribute (***IsCrypted***) that is attached to the field.

```
''' <summary>
''' My Field class
''' </summary>
Public Class MyField
    Inherits RepPP.Field

    ''' <summary>
    ''' contructor
    ''' </summary>
    ''' <param name="pobj"> C++ wrapper object</param>
    Public Sub New(ByVal pobj As Integer)
        MyBase.New(pobj)
    End Sub

    ''' <summary>
    ''' IsCrypted:      Returns whether or not the field is crypted
```

```
    ''' </summary>
    Public ReadOnly Property IsCrypted() As Boolean
        Get
            Dim bRetVal As Boolean
            Dim strValue As String

            strValue = Me.Attributes.GetValue("IsCrypted", "0")
            bRetVal = (strValue <> "0")
            Return bRetVal
        End Get
    End Property

End Class
```

Implementing a class factory

To define a class factory, you should create a class that implements the **RepPP.IObjFactory** interface. This interface contains one method, **CreateObject**. The method allows the creation of the custom object.

```
Public Class FieldFactory
    Implements RepPP.IObjFactory

    ''' <summary>
    ''' CreateObject        Create object
    ''' </summary>
    ''' <param name="type"> Type to create</param>
    ''' <param name="pobj"> Pointer to the C++ wrapper object</param>
    ''' <returns>
    ''' object instance
    ''' </returns>
    ''' <remarks></remarks>
    Public Function CreateObject(ByVal type As System.Type, _
                                 ByVal pobj As IntPtr) As _
                            RepPP.RepPPObject Implements RepPP.IObjFactory.CreateObject
        Dim objRetVal As RepPP.RepPPObject

        If type Is GetType(RepPP.FieldDef) Then
            objRetVal = New MyFieldDef(pobj)
        Else
            Throw New System.ApplicationException( _
                      "Error: cannot create an object of type " + type.Name)
        End If
        Return objRetVal
    End Function
End Class
```

Suppose that you want to create the **FieldDef** custom class only for fields of type **String**. The following code shows how to change the **FieldFactory** class:

```
Public Class FieldFactory
    Implements RepPP.IObjFactory
    ''' <summary>
    ''' Old field factory
    ''' </summary>
    Private m_factoryOld As RepPP.IObjFactory

    ''' <summary>
    ''' Constructor
    ''' </summary>
    Public Sub New()
        Dim app As RepPP.Application

        app = RepPP.Application.GetApplication()
        m_factoryOld = app.GetObjFactory(GetType(RepPP.FieldDef))
```

```
    End Sub

    ''' <summary>
    ''' CreateObject        Create object
    ''' </summary>
    ''' <param name="type"> Type to create</param>
    ''' <param name="pobj"> Pointer to the C++ wrapper object</param>
    ''' <returns>
    ''' object instance
    ''' </returns>
    ''' <remarks></remarks>
    Public Function CreateObject(ByVal type As System.Type, _
                                 ByVal pobj As IntPtr) As _
                    RepPP.RepPPObject Implements RepPP.IObjFactory.CreateObject
        Dim objRetVal As RepPP.RepPPObject
        Dim fldDef As RepPP.FieldDef

        If type Is GetType(RepPP.FieldDef) Then
            fldDef = CType(m_factoryOld.CreateObject(GetType(RepPP.FieldDef), pobj), _
                           RepPP.FieldDef)
            If fldDef.Type = RepPP.FieldType.sdFieldString Then
                objRetVal = New MyFieldDef(pobj)
            Else
                objRetVal = fldDef
            End If
        Else
            Throw New System.ApplicationException( _
                      "Error: cannot create an object of type " + type.Name)
        End If
        Return objRetVal
    End Function
End Class
```

Registering the class factory

To register the class factory, you should use the **RepPP.Application.SetObjFactory** method. This method should be called immediatly after the REP++ initialization to ensure that no other REP++ object exists.

```
Dim app As RepPP.Application

app = RepPP.Application.CreateFromRes()
app.SetObjFactory(GetType(RepPP.FieldDef), New FieldFactory())
```

Once registered, the class factory becomes responsible for creating the REP++ objects, and the REP++ **Application** will delegate the task of creating new instances to this class.