Every verifier has a *VerifierErrorMessageInfo* property which in turn has a *ResourceName* property. This name is automatically generated (and set) during code generation for attribute based verifiers. For example, note the autogenerated *ErrorMessageResourceNames* on these attributes:

```
[StringLengthVerifier(MaxValue=15, IsRequired=false, ErrorMessageResourceName="Employee_Country")]
public string Country {}
[RequiredValueVerifier(ErrorMessageResourceName="OrderDetail_Quantity")]
public short Quantity {}
```

```
<StringLengthVerifier(MaxValue:=15, IsRequired:=False, ErrorMessageResourceName:="Employee_Country")> _
Public ReadOnly Property Country() As String
End Property
<RequiredValueVerifier(ErrorMessageResourceName:="OrderDetail_Quantity")> _
Public ReadOnly Property Quantity() As Short
End Property
```

This resource name can also be set programmatically for any verifier defined in code.

The VerifierEngine has an *ErrorsResourceManager* property that can assigned to like this:

```
verifierEngine.ErrorsResourceManager = MyErrorMessages.ResourceManager;
```

```
verifierEngine.ErrorsResourceManager = MyErrorMessages.ResourceManager
```

where MyErrorMessages is the name of the strongly typed Designer.cs class generated by the .NET ResX editor. The *VerifierEngine* will look up each verifier's *ResourceName* in the registered *ErrorsResourceManager* in order to generate validation error messages.

When you set *ErrorsResourceManager*, you set the *ResourceManager* for *the entire application*, not just for this particular *VerifierEngine*.

Regarding localization of these resources, please see Microsoft's documentation for more information on this topic.

The *VerifierEngine* has a fallback *ResourceManager* it uses for system default error messages. These resource names may be overridden in the programmatically specified *ResourceManager*. The fallback manager ResX is shown below.

Most error messages contain substitution parameters; the meaning and order of these substitution parameters is defined by the type of verifier involved. For example:

- *{0} is required.*
- *{0} must be between {1} and {2} character(s).*

By simply including {0}, {1}, and {2} in any message, you can easily determine what parameters are used for that verifier.

Fallback ResourceManager ResX:

```
<?xml version="1.0" encoding="utf-8"?>
<root>
 <!--
   Microsoft ResX Schema

   Version 2.0

   The primary goals of this format is to allow a simple XML format
   that is mostly human readable. The generation and parsing of the
   various data types are done through the TypeConverter classes
   associated with the data types.

   Example:

   ... ado.net/XML headers & schema ...
   <resheader name="resmimetype">text/microsoft-resx</resheader>
   <resheader name="version">2.0</resheader>
   <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms, ...</resheader>
   <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms, ...</resheader>
   <data name="Name1"><value>this is my long string</value><comment>this is a comment</comment></data>
   <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
   <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
      <value>[base64 mime encoded serialized .NET Framework object]</value>
   </data>
   <data name="Icon1" type="System.Drawing.Icon, System.Drawing" mimetype="application/x-microsoft.net.object.bytearray.base64">
      <value>[base64 mime encoded string representing a byte array form of the .NET Framework object]</value>
      <comment>This is a comment</comment>
```

```
    </data>

    There are any number of "resheader" rows that contain simple
    name/value pairs.

    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.

    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:

    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.

    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-
msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0" msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
```

```
            <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
          </xsd:sequence>
          <xsd:attribute name="name" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:choice>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
<resheader name="resmimetype">
 <value>text/microsoft-resx</value>
</resheader>
<resheader name="version">
 <value>2.0</value>
</resheader>
<resheader name="reader">
 <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089</value>
</resheader>
<resheader name="writer">
 <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089</value>
</resheader>
<data name="VerifierAnd" xml:space="preserve">
 <value>{0} and {1}</value>
</data>
<data name="VerifierBetween" xml:space="preserve">
 <value>{0} must be between {1} and {2}</value>
</data>
<data name="VerifierBetweenDate" xml:space="preserve">
 <value>{0} must be between {1:d} and {2:d}</value>
</data>
<data name="VerifierBetweenStringLength" xml:space="preserve">
 <value>{0} must be between {1} and {2} character(s)</value>
</data>
<data name="VerifierInList" xml:space="preserve">
 <value>Value must be among the valid choices for {0}</value>
</data>
<data name="VerifierMaxDate" xml:space="preserve">
 <value>{0} must be on or before {1:d}</value>
</data>
<data name="VerifierMaxDateExclusive" xml:space="preserve">
 <value>{0} must be before {1:d}</value>
</data>
<data name="VerifierMaxStringLength" xml:space="preserve">
 <value>{0} cannot be longer than {1} characters(s)</value>
</data>
<data name="VerifierMaxValue" xml:space="preserve">
 <value>{0} must be less than or equal to {1}</value>
</data>
<data name="VerifierMaxValueExclusive" xml:space="preserve">
 <value>{0} must be less than {1}</value>
</data>
<data name="VerifierMinDate" xml:space="preserve">
 <value>{0} must be on or after {1:d}</value>
</data>
<data name="VerifierMinDateExclusive" xml:space="preserve">
 <value>{0} must be after {1:d}</value>
</data>
<data name="VerifierMinStringLength" xml:space="preserve">
 <value>{0} cannot be shorter than {1} character(s)</value>
</data>
<data name="VerifierMinValue" xml:space="preserve">
 <value>{0} must be greater than or equal to {1}</value>
</data>
<data name="VerifierMinValueExclusive" xml:space="preserve">
 <value>{0} must be greater than {1}</value>
</data>
<data name="VerifierNotRequired" xml:space="preserve">
 <value>{0} is not required</value>
</data>
```

```
  <data name="VerifierRequired" xml:space="preserve">
    <value>{0} is required</value>
  </data>
  <data name="VerifierValid" xml:space="preserve">
    <value>{0} must be a valid {1}</value>
  </data>
  <data name="VerifierValidPattern" xml:space="preserve">
    <value>{0} must be a valid {1} pattern</value>
  </data>
</root>
```