Working with Capture Collections in a .NET Environment

Application Note

Date	June 25, 2012
Applies To	Kofax Capture 8.0, 9.0, 10.0 Ascent Capture 7.x
Summary	This application note provides information regarding development using Capture product's collections in a .NET environment such as VB.NET or C#.
Revision	1.5

Overview

In the past, when developing Custom Modules, Workflow Agents, Release Scripts, Visual Basic 6 was typically the tool of choice. The internal components implemented by Capture products were COM-based, so it was relatively easy to iterate through a collection directly. However, the advent of .NET brought a new set of problems to developers. This article should hopefully address some of those issues.

NOTES:

This application note provides a workaround for accessing some of the collections in the Capture product. The API used here to access these collections is generally unpublished and thereby subject to change in future versions of the product.

This document targets primarily the Export Connector (formerly called Release Script) development API.

The Problem

Previously, in the VB 6 environment, one could easily iterate through a collection because the collection was a simple list containing Variant types:

```
Dim oVar As Variant
Dim str As String
For Each oVar In SetupData.BatchVariableNames
      str = oVar
      List1.AddItem str
Next oVar
```

Microsoft did away with Variants in .NET. Because of this, some of the Kofax Capture / Ascent Capture collections are now Object-based. The issue is compounded due to these collections not having any sort of “Count” property.

The Solution

Following are some examples of implementing IEnumerable and IEnumerator to iterate the COM-based collections in Capture.

A Word about Garbage Collection

We have seen problems using 'for…each' loops in .NET clients, and we have seen instances where the .NET Garbage Collector will free up the enumerator if the loop processing takes too long.

In our examples, we use a 'While' loop. Note the use of the "GC.KeepAlive" method placed at the end of each loop. Microsoft defines this method as:

> "References the specified object, making it ineligible for garbage collection from the start of the current routine to the point where this method is called."

In our examples, the "iMyEnum" object will be protected from the Garbage Collector from the beginning of the subroutine to the line containing the GC.KeepAlive statement.

The BatchVariableNames Collection

BatchVariableNames is a collection in the ReleaseSetupData object which contains a list of Object types. Implementing an IEnumerator, you can safely iterate through the collection in the following manner:

In VB.NET

```
If BVNames Is Nothing Then
    BVNames = New List(Of String)
    Dim iMyEnum As System.Collections.IEnumerator =
                                SetupDat.BatchVariableNames.GetEnumerator()
    iMyEnum.Reset()
    While iMyEnum.MoveNext()
        BVNames.Add(iMyEnum.Current.ToString())
    End While
    GC.KeepAlive(iMyEnum)
End If
Return BVNames
```

In C#

```
if (BVNames == null)
{
    BVNames = new List<string>();
    System.Collections.IEnumerator iMyEnum =
      ((System.Collections.IEnumerable)SetupDat.BatchVariableNames).GetEnumerator(
);
    iMyEnum.Reset();
    while (iMyEnum.MoveNext())
    {
        BVNames.Add((string)iMyEnum.Current);
    }
    GC.KeepAlive(iMyEnum);
}
return BVNames;
```

The ImageTypes Collection

NOTE: In a previous version of this document, we recommended using the Interop.IMAGESERVICELib.dll. This particular library has been replaced in Kofax Capture 9.0 and is no longer available. We therefore recommend utilizing the late bound method outlined here.

For this technique, we will need an object that represents an Image Type. Here is a class that satisfies the requirement. Listing 1 shows the ImageType class in VB.NET and Listing 2 is the same class written in C#.

```
Public Class ImageType

    Private m_oImageType As Object

    ' Constructor.
    ' oImageType = Type of the image
    Sub New(ByVal oImageType As Object)
m_oImageType = oImageType
    End Sub

    ' Get image description.
    Public ReadOnly Property Description() As String
        Get
            Dim typ As Type = m_oImageType.GetType()
            Return typ.InvokeMember("Description", System.Reflection.BindingFlags.GetProperty,
                                    Nothing, m_oImageType, Nothing).ToString()
        End Get
    End Property

    ' Get value indicates whether image is multi page or not.
    ' Returns True if image is multi page; otherwise, False
    Public ReadOnly Property MultiplePage() As String
        Get
            Dim typ As Type = m_oImageType.GetType()
            Return typ.InvokeMember("MultiplePage", System.Reflection.BindingFlags.GetProperty,
                                    Nothing, m_oImageType, Nothing).ToString()
        End Get
    End Property

    ' Get image type.
    Public ReadOnly Property Type() As Integer
        Get
            Dim typ As Type = m_oImageType.GetType()
            Return typ.InvokeMember("Type", System.Reflection.BindingFlags.GetProperty,
                                    Nothing, m_oImageType, Nothing)
        End Get
    End Property

    ' Return the default string for this class which is the Description.
    Public Overrides Function ToString() As String
        Return Description
    End Function

End Class
```

Listing 1

```
class ImageType
{
    private object m_oImageType;

    // Constructor.
    // oImageType = Type of the image
    public ImageType(object oImageType)
    {
        m_oImageType = oImageType;
    }

    // Get image description.
    public string Description
    {
        get
        {
            Type typ = m_oImageType.GetType();
            return (string)typ.InvokeMember("Description", System.Reflection.BindingFlags.GetProperty,
                                             null, m_oImageType, null);
        }
    }

    // Get value indicates whether image is multi page or not.
    // Returns True if image is multi page; otherwise, False
    public int MultiplePage
    {
        get
        {
            Type typ = m_oImageType.GetType();
            return (int)typ.InvokeMember("MultiplePage", System.Reflection.BindingFlags.GetProperty,
                                         null, m_oImageType, null);
        }
    }

// Get image type.
    public int Type
      {
        get
        {
            Type typ = m_oImageType.GetType();
            return (int)typ.InvokeMember("Type", System.Reflection.BindingFlags.GetProperty,
                                         null, m_oImageType, null);
        }
    }

    // Return the default string for this class which is the Description.
    public override string ToString()
    {
        return Description;
    }
}
```

Listing 2

To implement the class, we first get an enumerator for iterating the ImageTypes collection. The ImageType class directly corresponds to the ImageType object in the ReleaseSetupData ImageTypes collection. The "Current" object in the loop will be an ImageType object which is simply added to our generic list.

In VB.NET

```
If ITypes Is Nothing Then
    ITypes = New Generic.List(Of ImageType)

    Dim iMyEnum As System.Collections.IEnumerator =
                                        SetupDat.ImageTypes.GetEnumerator()
    iMyEnum.Reset()
    While iMyEnum.MoveNext()
        Dim oImageType As ImageType = New ImageType(iMyEnum.Current)
        ITypes.Add(oImageType)
    End While
    GC.KeepAlive(iMyEnum)
End If
Return ITypes
```

In C#

```
if (ITypes == null)
{
    ITypes = new List<ImageType>();

    System.Collections.IEnumerator iMyEnum =
   ((System.Collections.IEnumerable)SetupDat.ImageTypes).GetEnumerator();
    iMyEnum.Reset();
    while (iMyEnum.MoveNext())
    {
        ImageType oImageType = new ImageType(iMyEnum.Current);
        ITypes.Add(oImageType);
    }
    GC.KeepAlive(iMyEnum);
}
return ITypes;
```

The CustomProperties Collection

The CustomProperties collection is available in both the ReleaseSetupData and ReleaseData objects. At Release Setup time, you add objects of type Kofax.ReleaseLib.CustomProperty by using the Add method of the collection.

In VB.NET

```
SetupDat.CustomProperties.Add("MyCustom", "MyValue")
```

In C#

```
SetupDat.CustomProperties.Add("MyCustom", "MyValue");
```

At Release time, the collection and its objects become read-only and are accessed using a “string” key value. In C#, you need to pass a reference to an “object” type to the get_Item( ) method of the collection.

In VB.NET

```
Dim abc As String

abc = DocumentData.CustomProperties.Item(“MyCustom”).Value
```

In C#

```
object strKey = "MyCustom";
string abc;
abc = rdDocumentData.CustomProperties.get_Item(ref strKey).Value;
```

You can iterate through the collections using the collection in a while loop like this:

In VB.NET

```
Dim iMyEnum As System.Collections.IEnumerator =
               ReleaseData.CustomProperties.GetEnumerator()
While iMyEnum.MoveNext()
      Dim oCustProp As Kofax.ReleaseLib.CustomProperty =
                              CType(iMyEnum.Current, CustomProperty)
End While
GC.KeepAlive(iMyEnum)
```

In C#

```
System.Collections.IEnumerator iMyEnum =
                              ReleaseData.CustomProperties.GetEnumerator();
while (iMyEnum.MoveNext())
{
   Kofax.ReleaseLib.CustomProperty oCustProp =
                                          (CustomProperty)iMyEnum.Current;
}
GC.KeepAlive(iMyEnum);
```