Sample Workflow Agents

Application Note

Date	January 14, 2014
Applies To	Kofax Capture 8.0, 9.0, 10.0, 10.1 Ascent Capture 7.x
Summary	This application note provides a few sample Workflow Agents for the Capture 7.x, 8.0, 9.0 and 10.x products.
Revision	2.3

Contents

Overview

This application note provides a few sample Workflow Agents written in the C# language that demonstrate solutions to some typical scenarios.

Move Loose Pages Workflow Agent

This Workflow Agent checks if there are any loose pages in a Batch (i.e., pages that are not assigned to any Document). If there are, then a new Document is created and the loose pages are moved into that Document. **Listing 1**, displayed below, contains the entire source code for the project.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using Kofax.ACWFLib;
using Kofax.DBLiteOpt;

namespace MoveLoosePages
{
    [Guid("167BAE04-0127-4280-AA57-B199D255C97A")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("MoveLoosePages.LoosePages")]
    public class LoosePages : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref Kofax.ACWFLib.ACWorkflowData oWorkflowData)
        {
            // We only want to do the separation after Scan.
            if (oWorkflowData.CurrentModule.ID == "scan.exe")
            {
                // First get the Root & Batch elements.
                ACDataElement oRoot = oWorkflowData.ExtractRuntimeACDataElement(0);
                ACDataElement oBatch = oRoot.FindChildElementByName("Batch");

                // Get the Document collection element.
                ACDataElement oDocuments = oBatch.FindChildElementByName("Documents");
                if (oDocuments == null)
                {
                    oBatch.CreateChildElement("Documents");
                    oDocuments = oBatch.FindChildElementByName("Documents");
                }

                // Get the Pages element and collection from the Batch. These are Loose Pages.
                ACDataElement oPages = oBatch.FindChildElementByName("Pages");
                ACDataElementCollection oPageCol = oPages.FindChildElementsByName("Page");

                // Create a new Document element then iterate through the
                // Batch Pages collection moving each to the new Document.
                if (oPageCol.Count > 0)
                {
                    // Get the Document's Pages element.
                    ACDataElement oDoc = oDocuments.CreateChildElement("Document");
                    ACDataElement oDocPages = oDoc.FindChildElementByName("Pages");
                    if (oDocPages == null)
                    {
                        oDoc.CreateChildElement("Pages");
                        oDocPages = oDoc.FindChildElementByName("Pages");
                    }
                            foreach(ACDataElement oDocPage in oPageCol)
                    {
                        // When moving Pages into the Document, they are
                        // moved into the Document's Pages element.
                        try
                        {
                            oDocPage.MoveToBack(ref oDocPages);
                        }
                        catch (Exception ex)
                        {
                            oWorkflowData.ErrorText = ex.Message;
                        }
                    }
}}}}            }
```

Listing 1

The steps to move loose pages are as follows:

1. Create ACDataElements for the Root and Batch.
2. Check to see if there is a Document element and create one if there is not.
3. Check to see if there are any pages in the Batch's Pages collection (loose pages).
4. If there are loose pages, then create a new Document based on an existing Document Class in the Batch.
5. Find or create a Pages collection in the new Document.
6. Finally, iterate through the Pages collection in the Batch, moving each to the new Document's Pages collection.

Get the Data Type of All Index Fields in a Batch Workflow Agent

Consider that there is a requirement for obtaining the data type for all the Index Fields in a Batch. One reason could be to determine the data type for insertion into a database at some later point in the workflow. This can be done in a Workflow Agent, but requires using both the AscentCaptureRuntime and AscentCaptureSetup ACDataElements. These are defined in the following files:

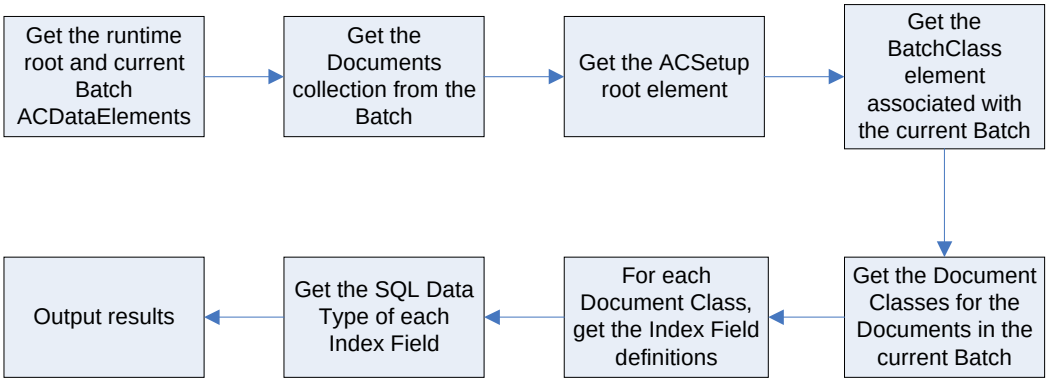
Ascent Capture 7.x

- C:\Program Files\AscentSS\CaptureSV\AcBatch.htm
- C:\Program Files\AscentSS \CaptureSV\AcSetup.htm

Kofax Capture 8.0, 9.0, and 10.x

- C:\Documents and Settings\All Users\Application Data\Kofax\CaptureSV\AcBatch.htm
- C:\Documents and Settings\All Users\Application Data\Kofax\CaptureSV\AcSetup.htm

The flow to obtain Index Field data types follows this general path:



The source code for the project is contained in **Listing 2**, located on the next two pages.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using Kofax.ACWFLib;
using Kofax.DBLiteOpt;
using System.Data.SqlClient;
using System.Windows.Forms;
using System.IO;

namespace WFAGetFieldInfo
{
    [Guid("ED1B171D-C72B-4152-972C-2B1F3C4E7F49")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("WFAGetFieldInfo.FieldInfo")]
    public class FieldInfo : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref Kofax.ACWFLib.ACWorkflowData oWorkflowData)
        {
            if (oWorkflowData.CurrentModule.ID.ToLower() == "scan.exe")
            {
                // Get the root & Batch elements.
                ACDataElement oRoot = oWorkflowData.ExtractRuntimeACDataElement(0);
                ACDataElement oBatch = oRoot.FindChildElementByName("Batch");

                // Get the Documents collection.
                ACDataElement oDoc = oBatch.FindChildElementByName("Documents");
                ACDataElementCollection oDocCol = oDoc.FindChildElementsByName("Document");

                // Get the SetupACDataElement root.
                ACDataElement oSetupRoot = oWorkflowData.ExtractSetupACDataElement(0);

                // Obtain the Batch Class definition of the current Batch.
                ACDataElement oCurrBatchClass = GetCurrentBatchClass(oSetupRoot, oBatch);

                // Instantiate a generic list to contain the data types for the
                // Document classes in this Batch. This list is of type DataType
                // which is defined in a class in this project.
                List<DataType> lstInfo = new List<DataType>();

                // Iterate through the Document Classes in the Batch Class.
                // Iterate through the Index Fields in the Document Classes.
                // Add the Field info to the list while ensuring unique objects.

                // Iterate through the Documents in this Batch. For each Document,
                // get it's Document Class and iterate through it's Index Fields
                // to get the field types.
                foreach (ACDataElement doc in oDocCol)
                {
                    ACDataElement oDocClass = GetDocumentClass(doc, oCurrBatchClass, oSetupRoot);
                    // Iterate through the Index Field collection and populate
                    // the generic list.
                    ACDataElement oIndexDefs = oDocClass.FindChildElementByName("IndexFieldDefinitions");
                    ACDataElementCollection oIndexDefsCol = oIndexDefs.FindChildElementsByName("IndexFieldDefinition");
                    foreach (ACDataElement oIndexDef in oIndexDefsCol)
                    {
                        DataType dt = new DataType();
                        dt.DocumentClass = oDocClass["Name"].ToString();
                        dt.IndexFieldName = oIndexDef["Name"].ToString();
                        dt.IndexFieldType = GetIndexType(oIndexDef["FieldTypeName"].ToString(), oSetupRoot);
                        lstInfo.Add(dt);
                    }
                }

                // Now that we have a generic List populated with the information
                // we need, we can now do with it what we want. In this case, we
                // are simply going to write the List items to a file.
                using (StreamWriter sw = File.AppendText(@"c:\WFATestFile.txt"))
                {
                    foreach (DataType dt in lstInfo)
                    {
                        sw.WriteLine("Document Class: " + dt.DocumentClass);
                        sw.WriteLine("Index Field   : " +  dt.IndexFieldName);
                        sw.WriteLine("Field Type    : " + dt.IndexFieldType);
                        sw.WriteLine("    ");
                    }
                }
            }
        }
        private ACDataElement GetCurrentBatchClass(ACDataElement SetupRoot, ACDataElement CurrBatch)
        {
            // Get the Batch Class definition element for the current Batch.
            ACDataElement oBatchClass = SetupRoot.FindChildElementByName("BatchClasses");
            ACDataElementCollection oBatchClassCol = oBatchClass.FindChildElementsByName("BatchClass");
```

Listing 2

```
            foreach (ACDataElement oBC in oBatchClassCol)
            {
                if (oBC["Name"].ToString() == CurrBatch["BatchClassName"])
                {
                    return oBC;
                }
            }
            return null;
        }

        private ACDataElement GetDocumentClass(ACDataElement doc, ACDataElement BatchClass, ACDataElement SetupRoot)
        {
            // Obtain the DocumentClass links collection.
            ACDataElement oDocLinks = BatchClass.FindChildElementByName("DocumentClassLinks");
            ACDataElementCollection oDocLinksCol = oDocLinks.FindChildElementsByName("DocumentClassLink");

            // First, iterate through the DocumentClassLinks collection
            // looking for the first link that applies to this Batch Class
            // and is the same as the current Document Class passed in.
            foreach (ACDataElement oDocLink in oDocLinksCol)
            {
                if (oDocLink["PublishedBatchDefID"].ToString() == BatchClass["PublishedBatchDefID"].ToString())
                {
                    // Get the Document Class that this link is pointing to by
                    // iterating through the DocumentClasses collection and
                    // searching for the specific DocumentClass element.
                    ACDataElement oDocClasses = SetupRoot.FindChildElementByName("DocumentClasses");
                    ACDataElementCollection oDocClassesCol = oDocClasses.FindChildElementsByName("DocumentClass");
                    foreach (ACDataElement oDocClass in oDocClassesCol)
                    {
                        if (oDocLink["DocumentClassName"].ToString() == oDocClass["Name"].ToString())
                        {
                            return oDocClass;
                        }
                    }
                }
            }
            return null;
        }

        private string GetIndexType(string name, ACDataElement SetupRoot)
        {
            // Get the FieldTypes collection.
            ACDataElement oFieldTypes = SetupRoot.FindChildElementByName("FieldTypes");
            ACDataElementCollection oFieldTypeCol = oFieldTypes.FindChildElementsByName("FieldType");

            // Search for a specific FieldType Name.
            foreach (ACDataElement FType in oFieldTypeCol)
            {
                if (FType["Name"].ToString() == name)
                {
                    int iType = int.Parse(FType["SqlType"].ToString());
                    return GetTypeString(iType);
                }
            }
            return "Unknown";
        }

        private string GetTypeString(int itype)
        {
            switch (itype)
            {
                case 1:
                    return "CHAR";
                case 2:
                    return "NUMERIC";
                case 3:
                    return "DECIMAL";
                case 4:
                    return "INTEGER";
                case 5:
                    return "SMALLINT";
                case 6:
                    return "FLOAT";
                case 7:
                    return "REAL";
                case 8:
                    return "DOUBLE";
                case 9:
                    return "DATETIME";
                case 12:
                    return "VARCHAR";
                default:
                    return "Unknown";
}}      }   }
```

Listing 2 (continued)

In this Workflow Agent (above), we are writing the output to a text file. However, this could be easily modified to output this information to a database, XML file, etc.

The first thing to do is obtain the AscentCaptureRuntime ACDataElement which represents the root object. From this element the current Batch ACDataElement is derived.

The next step is to obtain the Documents collection from the Batch.

We then obtain the AscentCaptureSetup ACDataElement which serves as the setup root object. We then pass this setup root object and the Batch object to the GetCurrentBatchClass method. This method gets the collection of BatchClasses from the setup root, and iterates through the collection while comparing the BatchClassNames until one is found that equals the Batch that was passed in to the method.

We decided to create the generic List at this time. The List is of type DataType which is a class defined containing three public member variables: DocumentClassName, IndexFieldName and IndexFieldType.

Regarding the Document Classes and their relationship to Batch Classes, because one Document Class can be used in different Batch Classes, the actual DocumentClasses collection exists in the setup root element. Likewise, the same is true for the FieldTypes collection, since a Field Type can be used in different Document Classes, Folder Classes, Batch Classes, etc.

When obtaining a Document Class, one must be careful to make sure the correct Document Class is being procured for the Batch Class in question. This is accomplished by first obtaining the DocumentClassLinks collection, then referencing the PublishedBatchClassDefID attribute in both the DocumentClassLink and BatchClass elements. If they are equal, then this is the correct Document Class. We have implemented the GetDocumentClass method to handle this issue. The method takes three ACDataElement parameters: Document, BatchClass and SetupRoot object. The DocumentLinks collection is obtained from the Batch Class, and then the collection is iterated through comparing Class Names with each Document Class in the DocumentClasses collection in the SetupRoot object. Once a match is found, the correct Document Class is returned from the method.

Now that we have our Document Class, the next thing to do is obtain the information about its Index Fields. For this, the IndexFieldDefinitions collection from the DocumentClass element is iterated through. In this loop we populate a generic List of DataType objects. We then iterate through the List and output some formatted information about the Index Fields to a text file.

Data Lookup Workflow Agent

Sometimes there is a need to perform a data lookup based on some Index Value or the like. Here is a Workflow Agent that implements the native .NET SqlClient namespace to perform a database lookup to obtain a sequence number. The number is assigned to an Index Field then incremented in the database. The source code for this project is in **Listing 3**, located on the next page.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using Kofax.ACWFLib;
using Kofax.DBLiteOpt;
using System.Data.SqlClient;

namespace DataLookup
{
    [Guid("9B3281D9-6C5B-4dee-9209-E55A23984CBE")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("DataLookup.SequenceNumber")]
    public class SequenceNumber : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref Kofax.ACWFLib.ACWorkflowData oWorkflowData)
        {
            if (oWorkflowData.CurrentModule.ID.ToLower() == "scan.exe")
            {
                // Set the Index Field Value
                ACDataElement root = oWorkflowData.ExtractRuntimeACDataElement(0);
                ACDataElement oBatch = root.FindChildElementByName("Batch");
                // Determine if this is the correct Batch Class
                if (oBatch["BatchClassName"].ToString() == "Test Numeric")
                {
                    using (SqlConnection oConn = new SqlConnection(
                          @"Data Source=mrobertson-wks\sqlexpress;Initial Catalog=TestData;Integrated Security=True"))
                    {
                        try
                        {
                            // Create a database connection and open the database
                            oConn.Open();
                            // Start a transaction process
                            SqlTransaction oTran = oConn.BeginTransaction();

                            ACDataElement oDocs = oBatch.FindChildElementByName("Documents");
                            ACDataElementCollection oDocCol = oDocs.FindChildElementsByName("Document");
                            foreach(ACDataElement oDoc in oDocCol)
                            {
                                // Connect to an SQL Server database, get the current sequence value,
                                // assign the value to an Index Field, increment the value then
                                // update the value in the SQL Server database.

                                int SeqNum = 0;

                                // Retrieve the value out of the database and set the Index Field to the value
                                SqlCommand oCmdSel = new SqlCommand("SELECT TOP 1 SequenceNumber FROM SequenceNumber", oConn);
                                oCmdSel.Transaction = oTran;
                                SeqNum = int.Parse(oCmdSel.ExecuteScalar().ToString());

                                // Set the sequence value into the Document Index Field
                                ACDataElement oIndexes = oDoc.FindChildElementByName("IndexFields");
                                ACDataElementCollection oIndexesCol = oIndexes.FindChildElementsByName("IndexField");
                                foreach (ACDataElement oIndex in oIndexesCol)
                                {
                                    if (oIndex["Name"] == "Number Field")
                                    {
                                        oIndex["Value"] = SeqNum.ToString();
                                        break;
                                    }
                                }

                                // Update the Sequence in the database with the incremented value.
                                SeqNum++;
                                SqlCommand oCmdUpdate = new SqlCommand("UPDATE SequenceNumber SET SequenceNumber=" +
                                                                       SeqNum.ToString(), oConn);

                                oCmdUpdate.Transaction = oTran;
                                oCmdUpdate.ExecuteNonQuery();
                            }
                            // Commit the transaction
                            oTran.Commit();
                        }
                        catch (Exception ex)
                        {
                            oWorkflowData.ErrorText = ex.Message;
                        }
                    } // using SqlConnection
                } // if BatchClassName
            } // if ModuleID = scan.exe
        }
    }
}
```

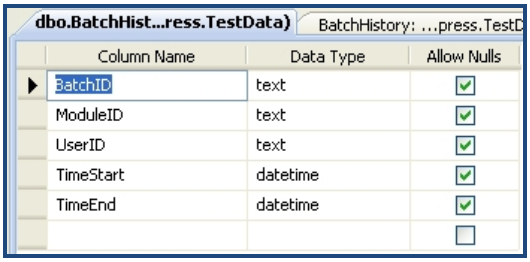
Listing 3

A Custom Batch History Workflow Agent

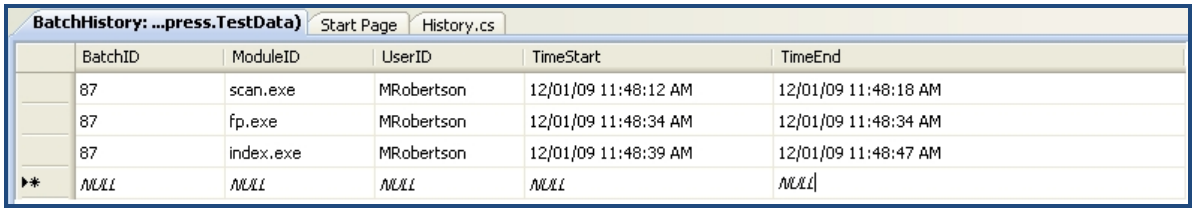
Let's assume there is a need to create a custom Batch History to your own database that is tailored to your specific needs. Here is a Workflow Agent that writes out the following values to a database:

- BatchID
- ModuleID
- UserID
- TimeStart
- Current Time & Date

We created this table in SQL Express.



And after running a Batch, this was the output in the table:



As the entries show, the Batch workflow contains the Scan, Recognition Server and Validation modules.

NOTE: Workflow Agents do not fire after Release.

Listing 4 (below) contains the code for this Workflow Agent. In the ProcessWorkflow event, we are trapping all of the trappable modules specified in this Batch Class. We get the last item in the BatchHistoryEntries collection and write out some values to a database.

We are using the current system Time and Date for the TimeEnd field because the Workflow Agent is fired as a part of the Batch closing process and the EndDateTime attribute in the BatchHistoryEntry has not been populated yet. This attribute is actually populated when the Batch is fully closed.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using Kofax.ACWFLib;
using Kofax.DBLiteOpt;
using System.Data.SqlClient;

namespace BatchHistory
{
    [Guid("E1D00C79-C8F6-4a70-849F-8E7FF7DFB7E8")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("BatchHistory.History")]
    public class History : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref Kofax.ACWFLib.ACWorkflowData oWorkflowData)
        {
            if (oWorkflowData.CurrentModule.ID == "scan.exe" ||
                oWorkflowData.CurrentModule.ID == "fp.exe" ||
                oWorkflowData.CurrentModule.ID == "index.exe")
            {
                // First drill down and get the Batch ACDataElement and the
                // BatchHistories collection.
                ACDataElement root = oWorkflowData.ExtractRuntimeACDataElement(0);
                ACDataElement oBatch = root.FindChildElementByName("Batch");
                ACDataElement oHistory = oBatch.FindChildElementByName("BatchHistoryEntries");
                ACDataElementCollection oHistoryCol = oHistory.FindChildElementsByName("BatchHistoryEntry");

                // Get the last entry in the collection. This should be for the current ModuleID.
                // However we will check to make sure.
                ACDataElement oHist = oHistoryCol[oHistoryCol.Count];
                if (oHist["ModuleID"].ToString() == oWorkflowData.CurrentModule.ID.ToString())
                {
                    // We found the element. Insert a record into the database.
                    using (SqlConnection oConn = new SqlConnection(
                                @"Data Source=mrobertson-wks\sqlexpress;Initial Catalog=TestData;Integrated Security=True"))
                    {
                        try
                        {
                            // Create a database connection and open the database
                            oConn.Open();
                            // Start a transaction process
                            SqlTransaction oTran = oConn.BeginTransaction();
                            string SQLInsert = "INSERT INTO BatchHistory (BatchID, ModuleID, UserID, TimeStart, TimeEnd)" +
                                               "VALUES ('" + oBatch["ExternalBatchID"].ToString() + "', '" +
                                                             oWorkflowData.CurrentModule.ID.ToString() + "', '" +
                                                             oHist["UserID"].ToString() + "', '" +
                                                             oHist["StartDateTime"].ToString() + "', '" +
                                                             DateTime.Now.ToString() + "')";
                            SqlCommand oCmdInsert = new SqlCommand(SQLInsert, oConn);
                            oCmdInsert.Transaction = oTran;
                            oCmdInsert.ExecuteNonQuery();
                            oTran.Commit();
                        }
                        catch (Exception ex)
                        {
                            // Put exception handling here.
}   }   }   }   }   }   }
```

Listing 4

Externally Programmable Required Field Workflow Agent

A customer wants to have required Index Fields in their Documents, but they do not want to configure them within the Document Class properties. They want to be able to configure specific fields externally in an XML file. After Recognition, if one or more required field is not populated, they want the Batch to be routed to Validation. If all Fields are populated, route the Batch to Release.

The steps that need to be performed in the Workflow Agent are as follows:

- The ProcessWorkflow event will execute code after Recognition only.
- Open and read the XML file and obtain the Index Field names.
- A generic List is populated with the Index Field names.
- For each Document in the Batch, drill down to the Index Fields.
- For each Index Field in the Document that has a name contained in the generic List, verify the Field has been populated with a value.
- If a target Field is not populated, exit the loop and route the Batch to Validation.
- If all target Fields are populated, route the Batch to Release.

The XML file will have the following format:

```
<?xml version="1.0" encoding="utf-8"?>
<IndexFields>
    <IndexField name="FirstName" />
    <IndexField name="LastName" />
</IndexFields>
```

For this Workflow Agent, we will be using a LINQ query to obtain the Index Field names from the XML file. Here is the LINQ query:

```
IEnumerable<XElement> ireqFields = (from f in xFile.Element("IndexFields").Elements("IndexField")
select f);
```

The complete source code for the Workflow Agent is in **Listing 5**, beginning on the next page.

NOTE: Visual Studio 2008 (.NET 3.5 or higher) is required to use the LINQ libraries.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.Windows.Forms;
using Kofax.ACWFLib;
using Kofax.DBLiteOpt;

namespace RequiredFieldsWFA
{
    [Guid("DA01B20B-2610-4f7d-8B3D-BB6FDD82E31F")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("RequiredFieldsWFA.RequiredFields")]
    public class RequiredFields : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref Kofax.ACWFLib.ACWorkflowData oWorkflowData)
        {
            ACDataElement oRoot = null;
            ACDataElement oBatch = null;
            ACDataElement oDocs = null;
            ACDataElementCollection oDocsCol = null;
            ACDataElement oIndexes = null;
            ACDataElementCollection oIndexesCol = null;

            try
            {
                // This code only executes after Recognition.
                if (oWorkflowData.CurrentModule.ID.ToLower() == "fp.exe")
                {
                    string sXMLPath = Path.Combine(Application.StartupPath, "RequiredFields.xml");

                    using (Microsoft.Win32.RegistryKey rk = Microsoft.Win32.Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Kofax
Image Products\Ascent Capture\3.0", false))
                    {
                        sXMLPath = Path.Combine(rk.GetValue("ServerPath").ToString(), "RequiredFields.xml");
                    }

                    List<string> ReqFieldsList = new List<string>();

                    // Make sure that the XML file exists.
                    if (!File.Exists(sXMLPath))
                    {
                        throw new ApplicationException("The RequiredFields.xml file was not present.");
                    }

                    // Load up the XML file and populate the generic List
                    // with the required Index Field names.
                    XDocument xFile = XDocument.Load(sXMLPath);
                    XElement reqFields = xFile.Element("IndexFields");

                    // This LINQ query gets all the Field names from the
                    // IndexFields top element.
                    IEnumerable<XElement> ireqFields = (from f in xFile.Element("IndexFields").Elements("IndexField")
                                                        select f);
                    foreach (XElement x in ireqFields)
                    {
                        ReqFieldsList.Add(x.Attribute("name").Value.ToString());
                    }

                    // Get the ACDataElements
                    oRoot = oWorkflowData.ExtractRuntimeACDataElement(0);
                    oBatch = oRoot.FindChildElementByName("Batch");
                    oDocs = oBatch.FindChildElementByName("Documents");
                    oDocsCol = oDocs.FindChildElementsByName("Document");

                    bool bNotPopulated = false;

                    // Iterate through the Documents
                    foreach (ACDataElement oDoc in oDocsCol)
                    {
                        // Iterate the Index Fields in the Document
                        // If an unpopulated required field is detected,
                        // set the flag and exit the loops.
                        oIndexes = oDoc.FindChildElementByName("IndexFields");
                        oIndexesCol = oIndexes.FindChildElementsByName("IndexField");
                        foreach (ACDataElement oIndex in oIndexesCol)
                        {
                            if (ReqFieldsList.Exists(ap => ap == oIndex["Name"].ToString()))
                            {
                                bNotPopulated = oIndex["Value"].ToString().Trim() == "" ? true : false;
                            }
                            Marshal.ReleaseComObject(oIndex);
                            if (bNotPopulated) break;
```

Listing 5

```
                        }
                        Marshal.ReleaseComObject(oDoc);
                        if (bNotPopulated) break;
                    }

                    // Iterate through the possible Modules for this
                    // Batch Class definition. Route to Release.
                    if (!bNotPopulated)
                    {
                        ACWorkflowModules oPossibleModules = oWorkflowData.PossibleModules;
                        foreach (ACWorkflowModule oModule in oPossibleModules)
                        {
                            if (oModule.ID.ToLower() == "release.exe")
                            {
                                ACWorkflowModule tMod = oModule;
                                oWorkflowData.set_NextModule(ref tMod);
                                break;
                            }
                            Marshal.ReleaseComObject(oModule);
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Application error: " + ex.Message);
            }
            finally
            {
                // Release the COM objects.
                if (oIndexesCol != null)
                {
                    Marshal.ReleaseComObject(oIndexesCol);
                    oIndexesCol = null;
                }
                if (oIndexes != null)
                {
                    Marshal.ReleaseComObject(oIndexes);
                    oIndexes = null;
                }
                if (oDocsCol != null)
                {
                    Marshal.ReleaseComObject(oDocsCol);
                    oDocsCol = null;
                }
                if (oDocs != null)
                {
                    Marshal.ReleaseComObject(oDocs);
                    oDocs = null;
                }
                if (oBatch != null)
                {
                    Marshal.ReleaseComObject(oBatch);
                    oBatch = null;
                }
                if (oRoot != null)
                {
                    Marshal.ReleaseComObject(oRoot);
                    oRoot = null;
                }
            }
        }
    }
}
```

Listing 5 (continued)

Programmatically Set a Batch Field

There is a need to set a common value into all Documents in a specific Batch. This value will need to be set automatically, possibly from a database lookup, a date value, from a file, etc.

This can easily be accomplished in a Workflow Agent. For this example, we are going to set a Batch Field value to the current Date & Time. This value will be set in a Field within all Documents of the Batch. Follow these steps:

- Create a Batch Class containing a Batch Field. In this case we named the Field “TestValue”.
- Make the Batch Field hidden so that the user cannot set it at Batch creation.
- In all Document Classes in this Batch Class, map the Default value to “{$TestValue}”, which is the Batch Field.
- Create a Workflow Agent to set the Batch Field value (see **Listing 6**).

```
using System;
using System.Collections.Generic;
using System.Text;
using Kofax.ACWFLib;
using Kofax.DBLite;
using Kofax.DBLiteOpt;
using System.Runtime.InteropServices;

namespace WFACopyBatchField
{
    [Guid("FEB73FE4-FBAE-46db-B25E-E4597727F0C6")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("WFACopyBatchField.BatchField")]
    public class BatchField : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref Kofax.ACWFLib.ACWorkflowData oWorkflowData)
        {
            // This WFA processes after the Recognition Module.
            if (oWorkflowData.CurrentModule.ID.ToLower() == "fp.exe")
            {
                // Retrieve the Batch element.
                ACDataElement root = oWorkflowData.ExtractRuntimeACDataElement(0);
                ACDataElement oBatch = root.FindChildElementByName("Batch");

                // Retrieve the Batch Field to copy. The Field name is hard
                // coded because this WFA will only be used with this Batch
                // Class. However, we still check the Batch Class name.
                if (oBatch["BatchClassName"] == "TEST")
                {
                    // Retrieve the BatchField element with the Name attribute = TestValue.
                    ACDataElement oFields = oBatch.FindChildElementByName("BatchFields");
                    ACDataElement oField = oFields.FindChildElementByAttribute("BatchField", "Name", "TestValue");
                    oField["Value"] = DateTime.Now.ToString();
                }
            }
        }
    }
}
```

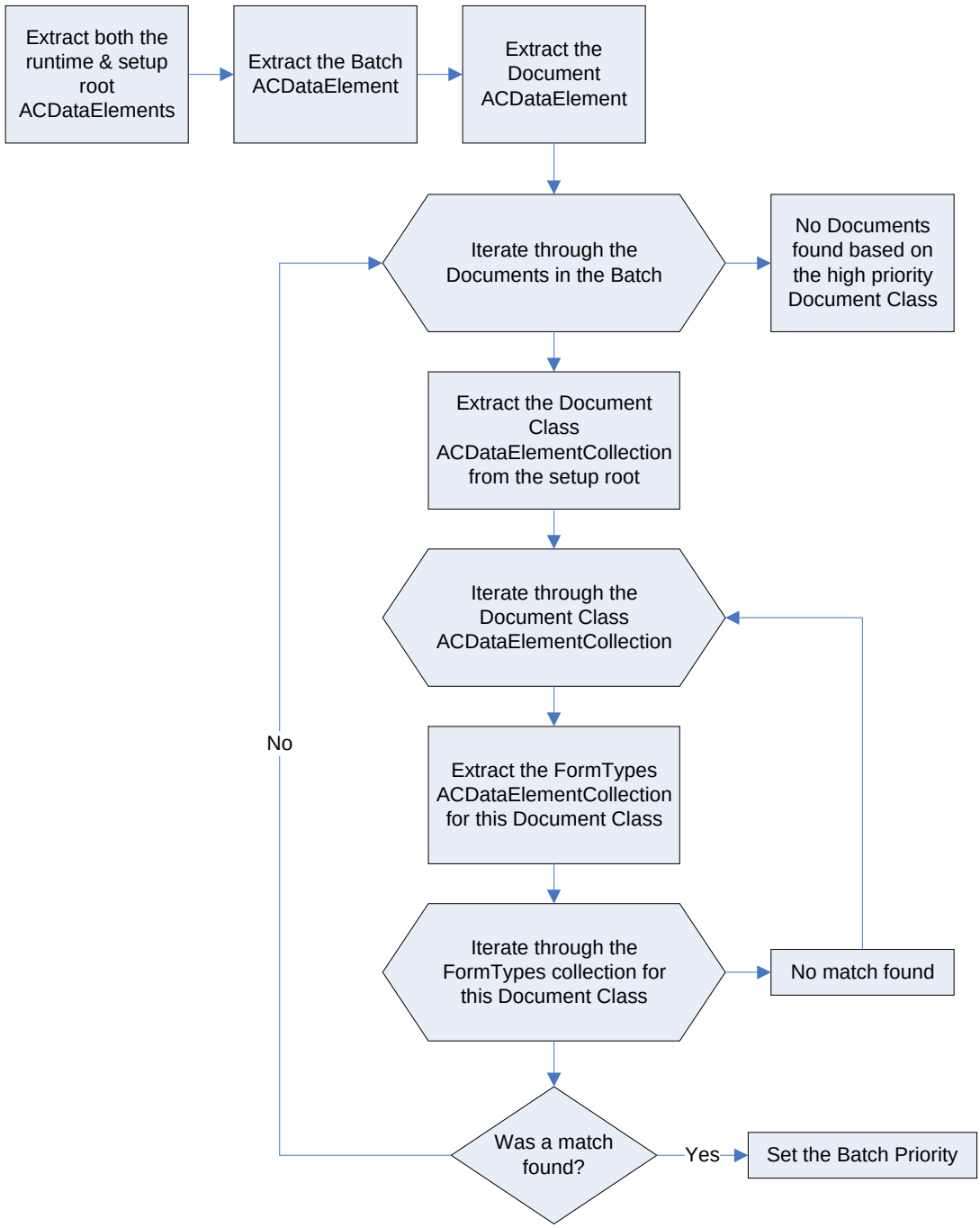
Listing 6

Note the use of the FindChildElementByName( ) method. This method takes three parameters. In this case, we are retrieving the “BatchField” element with the “Name” attribute equaling “TestValue”. The “Value” attribute of this element is then set to the current system Date and Time.

This value will be propagated to all Documents that have a Field which was properly mapped to this Batch-level Field.

Programmatically Set a Batch Priority

Related to the previous topic, “Programmatically Set a Batch Field,” here is a Workflow Agent that sets the Priority of a Batch based on some criteria. For this example, let’s assume that the Batch Class this Workflow Agent is added to has several Document Classes. One of the Document Classes is to have an importance that is placed at a higher value than the other Document Classes in this Batch Class. The Workflow Agent will analyze the Documents in the current Batch. If a Document exists based on the Document Class with the higher importance, the Batch Priority attribute will be changed to “1”.



The procedure for obtaining the Document Class names of the Documents in a Batch follows this sequence (see **Listing 7**).

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using Kofax.ACWFLib;
using Kofax.DBLiteOpt;

namespace SetPriority
{
    [Guid("5B9A4F33-4FAD-4288-9FEC-7897777F2FB7")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("SetPriority.SetPriority")]
    public class SetPriority : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref ACWorkflowData oWorkflowData)
        {
            // This executes after the Recognition Module.
            if (oWorkflowData.CurrentModule.ID.ToLower() == "fp.exe")
            {
                 //global::System.Windows.Forms.MessageBox.Show("Test3");
                 ACDataElement root = oWorkflowData.ExtractRuntimeACDataElement(0);
                 ACDataElement oBatch = root.FindChildElementByName("Batch");

                 if (IsPriorityDocument(oWorkflowData))
                 {
                    oBatch["Priority"] = "1";
                 }
            }
        }

        private bool IsPriorityDocument(ACWorkflowData oWorkflowData)
        {
            // We will need both the Runtime and the SetupRuntime ACDataElements.
            ACDataElement oRoot = oWorkflowData.ExtractRuntimeACDataElement(0);
            ACDataElement oSetup = oWorkflowData.ExtractSetupACDataElement(0);

            ACDataElement oBatch = oRoot.FindChildElementByName("Batch");
            ACDataElement oDocument = oBatch.FindChildElementByName("Documents");
            ACDataElementCollection oDocColl = oDocument.FindChildElementsByName("Document");

            // for each Document in this current Batch...
            foreach (ACDataElement oDoc in oDocColl)
            {
                // Get the setup info of the Document's Class settings...
                ACDataElement oDocClass = oSetup.FindChildElementByName("DocumentClasses");
                ACDataElementCollection oDocClassColl = oDocClass.FindChildElementsByName("DocumentClass");

                // For each Document Class in the Setup root...
                foreach (ACDataElement oSetupDoc in oDocClassColl)
                {
                    ACDataElement oFormType = oSetupDoc.FindChildElementByName("FormTypes");
                    ACDataElementCollection oFormTypeColl = oFormType.FindChildElementsByName("FormType");

                    // Iterate through the FormTypes collection.
                    foreach (ACDataElement oForm in oFormTypeColl)
                    {
                        if (oForm["Name"] == oDoc["FormTypeName"])
                        {
                            if (oSetupDoc["Name"] == "MyClassName")
                            {
                                // Document Class Name found.
                                return true;
                            }
                            break;
                        }
                    }
                }
            }
            return false;
        }
    }
}
```

Listing 7

Here we are setting the Batch Priority when a Document Class named “MyClassName” exists in the Batch. Instead of hard coding the name, an external value might be used from a database, text file or XML file.

Another method for trapping a Document Class Name is to implement a hidden field in the Document Class, then setting its default value to “{Document Class Name}”. The Workflow Agent could then simply look at the Index value and set the priority attribute accordingly.

A Batch Totaling Workflow Agent

Here is a simple Workflow Agent that maintains a Batch Field to contain a summation of a specific Field in the Documents of the Batch. After Validation, the Documents in the Batch are iterated through. For each Document, the value of the "Sum" field is accumulated into the "Total" Field in the Batch. See **Listing 8** for the source code.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using Kofax.DBLiteOpt;
using Kofax.ACWFLib;

namespace WFADocTotal
{
    [Guid("35E8AAE9-45D0-43d7-B75C-A89EAAC26846")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("WFADocTotal.DocTotal")]
    public class DocTotal : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref ACWorkflowData oWorkflowData)
        {
            // This Workflow Agent processes after the Validation Module.
            if (oWorkflowData.CurrentModule.ID == "index.exe")
            {
                Decimal dTotal = 0;
                ACDataElement root = oWorkflowData.ExtractRuntimeACDataElement(0);
                ACDataElement oBatch = root.FindChildElementByName("Batch");
                ACDataElement oDocument = oBatch.FindChildElementByName("Documents");
                ACDataElementCollection oDocCol = oDocument.FindChildElementsByName("Document");

                foreach (ACDataElement oDoc in oDocCol)
                {
                    // We are doing a blind Try..Catch here because there
                    // could be a Document that may not have an Index Field
                    // with a Name that equals "Sum".
                    try
                    {
                        ACDataElement oIndexes = oDoc.FindChildElementByName("IndexFields");
                        ACDataElement oSumField = oIndexes.FindChildElementByAttribute("IndexField", "Name", "Sum");
                        dTotal += Decimal.Parse(oSumField["Value"].ToString());
                    }
                    catch {}
                }
                GC.KeepAlive(oDocCol);

                ACDataElement oBFields = oBatch.FindChildElementByName("BatchFields");
                ACDataElement oBField = oBFields.FindChildElementByAttribute("BatchField", "Name", "Total");
                oBField["Value"] = dTotal.ToString();
            }
        }
    }
}
```

Listing 8

Note the use of the "FindChildElementByAttribute on the IndexFields element. This is much simpler than iterating through all the Index Fields looking for a match on the Name attribute.

Parsing the Original File Name

A customer wants to use the file name of the first page in a Document and parse out its individual parts into a series of Index Fields.

For this example, we are going to configure a Document Class with three Index Fields named Part1, Part2 and Part3. To keep things simple, all three Field Types are VARCHAR with a length of 10 characters.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using Kofax.ACWFLib;
using Kofax.DBLiteOpt;
using System.Runtime.InteropServices;

namespace WFAGetOriginalFilename
{
    [Guid("AF07DF16-36F7-44c3-BF40-6BC293EA1D6B")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("GetOriginalFilename.GetOriginalFilename")]
    public class GetOriginalFilename : _IACWorkflowAgent
    {
        public void ProcessWorkflow(ref Kofax.ACWFLib.ACWorkflowData oWorkflowData)
        {
            if (oWorkflowData.CurrentModule.ID.ToLower() == "scan.exe")
            {
                ACDataElement root = oWorkflowData.ExtractRuntimeACDataElement(0);
                ACDataElement oBatch = root.FindChildElementByName("Batch");

                ACDataElement oDocuments = oBatch.FindChildElementByName("Documents");
                ACDataElementCollection oDocCol = oDocuments.FindChildElementsByName("Document");
                foreach (ACDataElement oDoc in oDocCol)
                {
                    ACDataElement oPages = oDoc.FindChildElementByName("Pages");
                    ACDataElementCollection oPagesCol = oPages.FindChildElementsByName("Page");
                    string[] BaseStr = oPagesCol[1]["OriginalFileName"].ToString().Split('_');
                    ACDataElement oIndexes = oDoc.FindChildElementByName("IndexFields");
                    ACDataElement oPart1 = oIndexes.FindChildElementByAttribute("IndexField", "Name", "Part1");
                    oPart1["Value"] = BaseStr[0];
                    ACDataElement oPart2 = oIndexes.FindChildElementByAttribute("IndexField", "Name", "Part2");
                    oPart2["Value"] = BaseStr[1];
                    ACDataElement oPart3 = oIndexes.FindChildElementByAttribute("IndexField", "Name", "Part3");
                    oPart3["Value"] = BaseStr[2];
                }
                GC.KeepAlive(oDocCol);
            }
        }
    }
}
```

Listing 9

For our example, the filenames will be three strings concatenated together, separated with underscore characters, (e.g., NAME_ABC_12345.tif). The Workflow Agent iterates through all Documents in the Batch. For each Document, the first Page in the Pages collection is acquired and its “OriginalFilename” attribute is read and parsed using the Split method. The string values are then written into the three Index Fields, Part1, Part2 and Part3 respectively.

Making a Batch Field Read-only

A customer has a process in place where a Batch Field is used at creation time to contain some information which should not be tampered with later. For example, a numeric value might be entered containing the expected number of Documents or Pages in the Batch. Then, in some later process this value could be checked against the actual number of Documents or Pages. If the numbers differ, then handle the Batch per the specific business requirements. However, an issue arises in that Batch Fields can be easily modified in the Batch Manager by opening the Batch Properties dialog.

This Workflow Agent restricts this behavior by writing the initial value to the Batch Custom Properties. If a user attempts to change the value, a message will be displayed and the Batch Field changed back to its original value.

```
using System;
using System.Runtime.InteropServices;
using Kofax.DBLiteOpt;
using Kofax.ACWFLib;
using System.Windows.Forms;

namespace WFASaveBatchValue
{
    [Guid("F027B944-F3D1-473B-929A-13FE8E88BFC6")]
    [ClassInterface(ClassInterfaceType.None)]
    [ProgId("WFASaveBatchValue.SaveBatchValue")]
    public class SaveBatchValue : IACWorkflowAgent
    {
        public void ProcessWorkflow(ref ACWorkflowData oWorkflowData)
        {
            string ModuleID = oWorkflowData.CurrentModule.ID.ToLower();

            if (ModuleID == "scan.exe" || ModuleID == "ops.exe")
            {
                ACDataElement oRoot = oWorkflowData.ExtractRuntimeACDataElement(0);
                ACDataElement oBatch = oRoot.FindChildElementByName("Batch");
                ACDataElement oFields = oBatch.FindChildElementByName("BatchFields");
                ACDataElement oField = oFields.FindChildElementByAttribute("BatchField", "Name", "TestValue");

                // Check if the Batch Field has already been set by checking
                // if the Custom Storage String was previously set.
                try
                {
                    if (oWorkflowData.BatchCustomStorageString["TestValue"].Trim() == "")
                    {
                        oWorkflowData.set_BatchCustomStorageString("TestValue", oField["Value"].ToString());
                    }
                    else
                    {
                        // If the value was previously set then display a message
                        // and reset the Batch Field to its previous value.
                        if (oWorkflowData.BatchCustomStorageString["TestValue"] != oField["Value"])
                        {
                            MessageBox.Show("This Batch Field cannot be changed once the Batch has been created.", "",
                                            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                            oField["Value"] = oWorkflowData.BatchCustomStorageString["TestValue"];
                        }
                    }
                }
                catch
                {
                    // We will probably enter this exception when attempting
                    // to set or Trim a value that doesn't exist yet.
                    oWorkflowData.set_BatchCustomStorageString("TestValue", oField["Value"].ToString());
                }
            }
        }
    }
}
```

Listing 10

When the Batch is created, either in Scan or the Batch Manager, as the Batch closes, the Workflow Agent writes the Batch Field to the Batch Custom Properties. If a user attempts to change the value via the Batch Properties in the Batch Manager, the Workflow Agent will again run, this time checking the changed Batch Field against the value stored in the Custom Properties. If the two values differ, then a message is displayed and the Batch Field is automatically reset back to its original value.