Kofax Import Connector

Developer's Guide

Version: 2.6.0

Date: 2017-05-31

Table of Contents

Overview

Introduction

This guide contains additional information about the interfaces of Kofax Import Connector, including:

- Description of the functions of the Message Connector Web Services Interface
- Information about the Web Services Sample Client
- Description of the Scripting Interface
- Custom Storage Strings

This guide assumes that you have a thorough understanding of Windows standards, applications, and interfaces. It also assumes that you have a thorough understanding of web services and Kofax Capture.

This guide is for developers who are intend to create a custom web service application or scripts for customizing Kofax Import Connector.

Related Documentation

In addition to this Kofax Import Connector Developer's Guide, the following additional documentation is available:

- Kofax Import Connector Installation Guide
- Kofax Import Connector Administrator's Guide
- Message Connector Help
- KC Plug-In Help
- Release notes

Help

The online Help systems included in Kofax Import Connector provide online assistance for system administrators and operators alike. You can access online Help from any application window by clicking Help.

Release Notes

Late-breaking product information is available from release notes. You should read the release notes carefully, as they contain information that may not be included in other Kofax Import Connector documentation.

Chapter 1

Message Connector Web Services Interface

Message Connector can act as an HTTP/HTTPS server that accepts standard web-service calls from any customer application. The interface is described by a WSDL file that can be imported by commonly used development tools (e.g. Microsoft VisualStudio):

http://{message-connector}:{port}/file/import.wsdl

The web service interface provides functions for importing documents to Kofax Capture. All other functions used by the internal web-service interface (for example, access to Web UI, get/view messages, etc.) are not supported by this HTTP server. You can safely use it in low-trusted networks or even the Internet.

Message Connector also provides several functions for use with Kofax Monitor. These functions are described in chapter Web Services Interface for Kofax Monitor.

Conformance to Standards

The web service interface is based on the Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000 (http://www.w3.org/TR/2000/NOTE-SOAP-20000508/).

The provided Web Service Description Language (WSDL) files use Version 1.1, W3C Note 15 March 2001 (http://www.w3.org/TR/wsdl).

All web service calls use the "document-literal" combination of SOAP binding style and data encoding. For a discussion of "document-literal" versus "rpc-encoded" models, see for example:

- http://java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns/
- http://msdn.microsoft.com/library/en-us/dnwebsrv/html/rpc_literal.asp
- http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/

The HTTP layer implements HTTP/1.1 as defined in RFC 2616. See http://www.w3.org/Protocols/rfc2616/rfc2616.html.

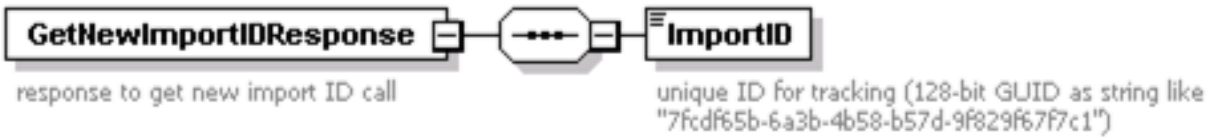
The HTTPS implementation is based on the OpenSSL Project, an open source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. OpenSSL version 1.0.1h from 5 June 2014 is used. See http://www.openssl.org/.

GetNewImportID Function

This function is used optionally to get a unique ImportID for a subsequent Import call. The function returns a 128bit GUID. If the client has any other method to generate a GUID it can be used as well.

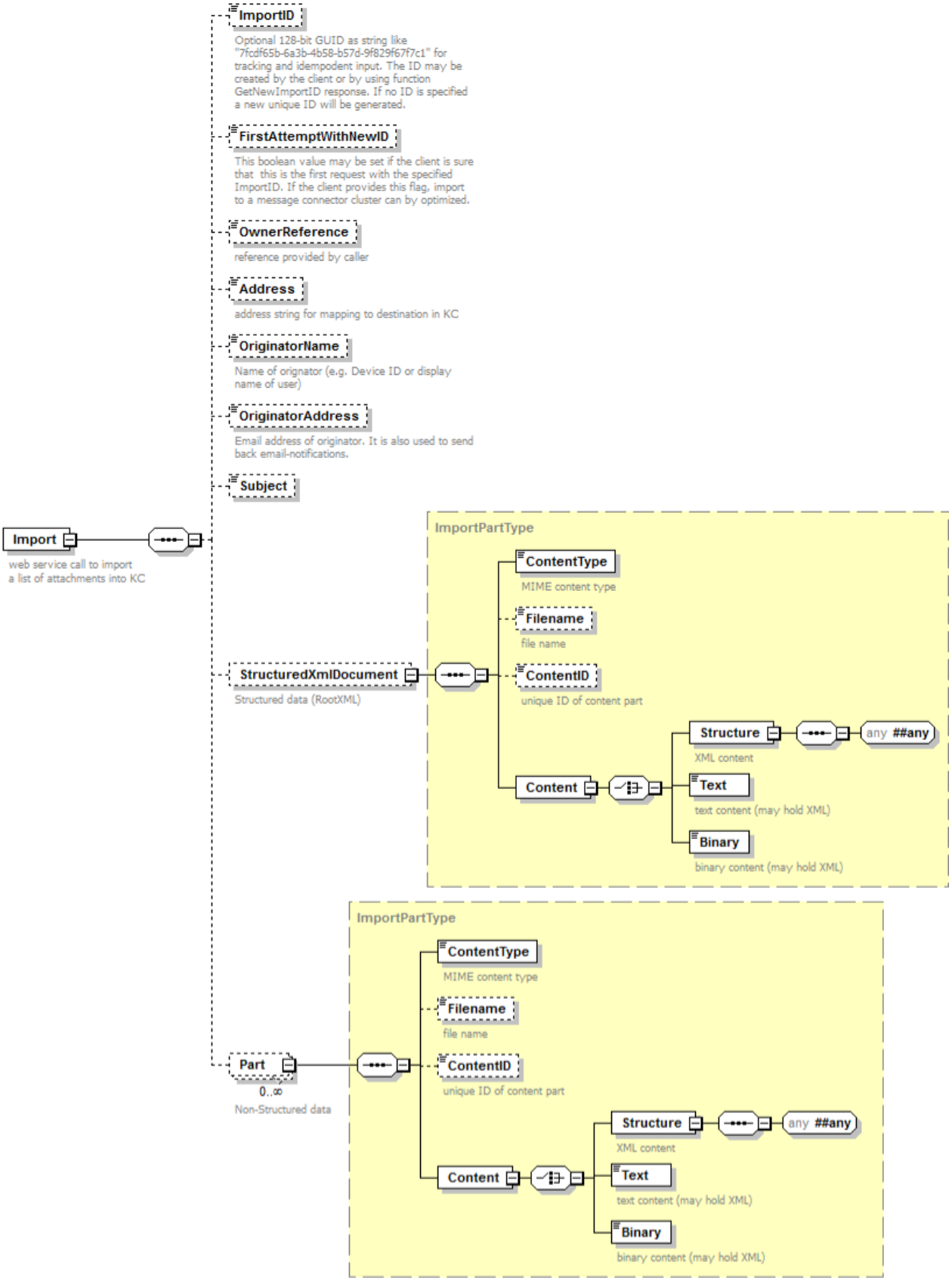
Import may also be called without specifying an ImportID, but knowing the ImportID in advance allows idempotent input e.g. in case of retries after Import failed without returning a response.

The GetNewImportID function has no parameters and returns a response as shown below:



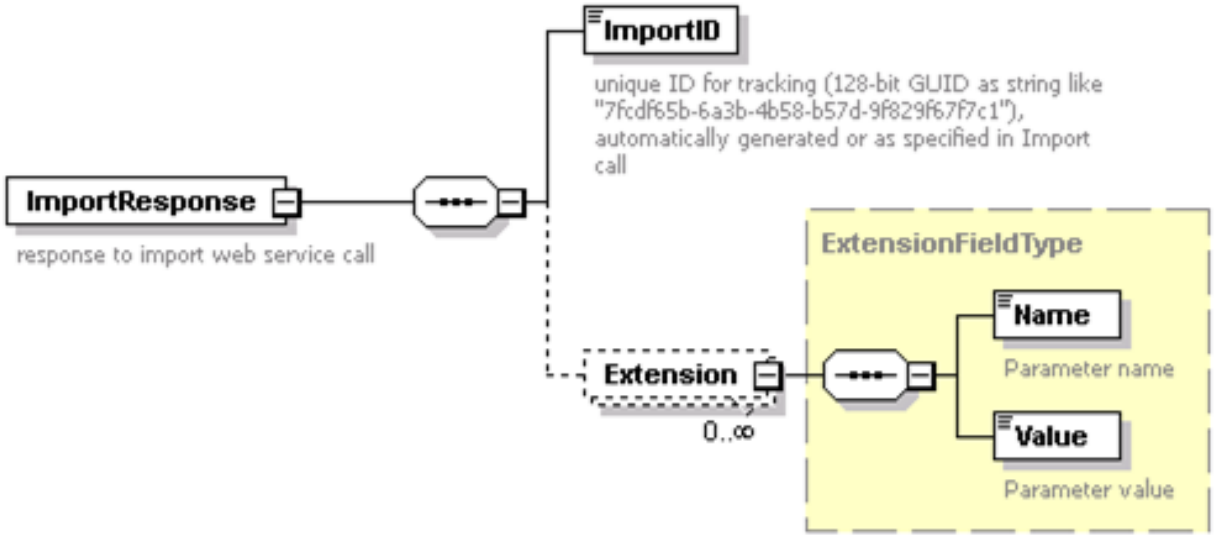
Import Function

This function accepts a list of attachments that are processed like an email. The input schema is shown below:


ImportID
Optional 128-bit GUID as string like "7fcdf65b-6a3b-4b58-b57d-9f829f67f7c1" for tracking and idempodent input. The ID may be created by the client or by using function GetNewImportID response. If no ID is specified a new unique ID will be generated.
FirstAttemptWithNewID
This boolean value may be set if the client is sure that this is the first request with the specified ImportID. If the client provides this flag, import to a message connector cluster can by optimized.
OwnerReference
reference provided by caller
Address
address string for mapping to destination in KC
OriginatorName
Name of orignator (e.g. Device ID or display name of user)
OriginatorAddress
Email address of originator. It is also used to send back email-notifications.
Subject
Import
web service call to import a list of attachments into KC
ImportPartType
ContentType
MIME content type
Filename
file name
StructuredXmlDocument
Structured data (RootXML)
ContentID
unique ID of content part
Structure
XML content
any ##any
Content
Text
text content (may hold XML)
Binary
binary content (may hold XML)
ImportPartType
ContentType
MIME content type
Filename
file name
Part
0..∞
Non-Structured data
ContentID
unique ID of content part
Structure
XML content
any ##any
Content
Text
text content (may hold XML)
Binary
binary content (may hold XML)

If the input should be handled as an XML message, StructuredXmlDocument must be provided with the content of the root-XML. Both StructuredXmlDocument and all other parts can be provided as (base-64 encoded) binary, (escaped) text, or as XML presentation.

If the message is accepted by the storage the following success response is returned:



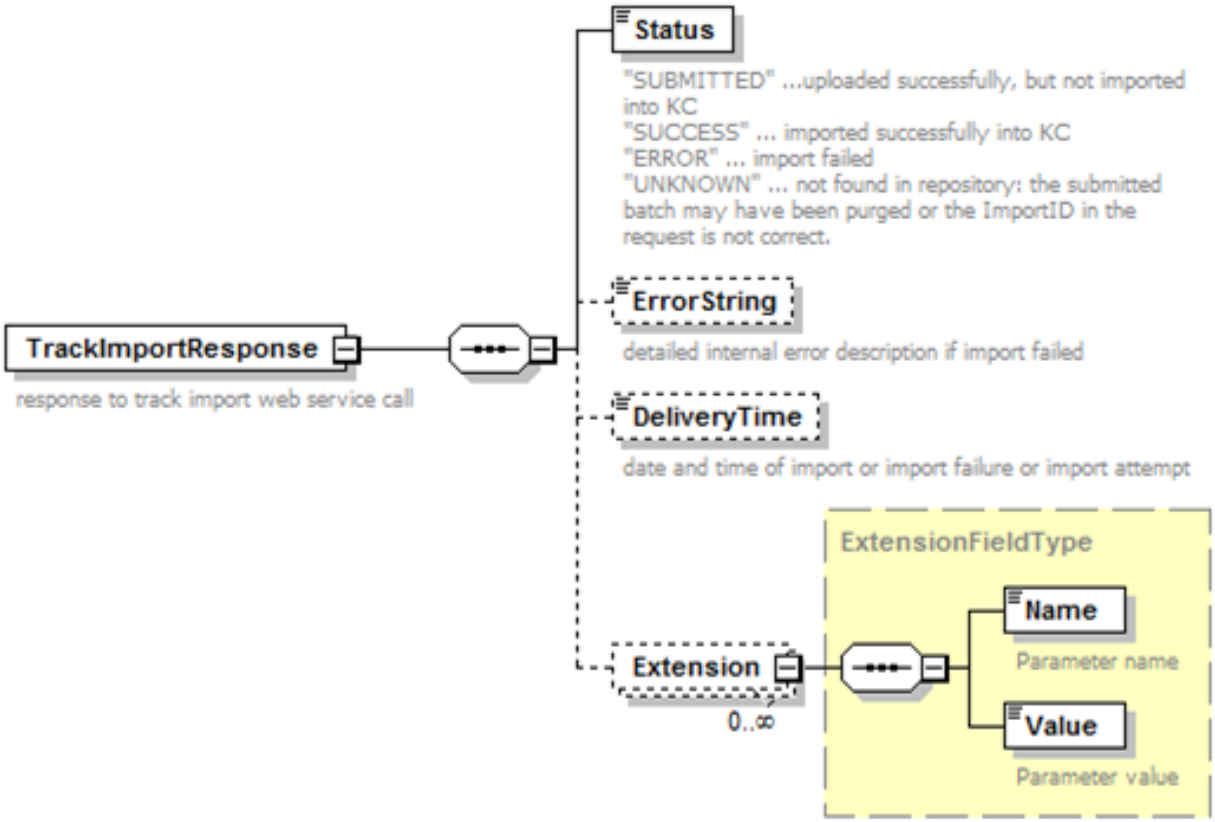
The ImportID returns the internal GUID of the message which may be used for tracking the status with function TrackImport. The Extension values are reserved for future extensions.

If the message is not accepted a standard SOAP fault object is returned.

TrackImport Function

This function returns the status of a message that has been previously imported with function Import.

On success, the function returns the status of the message as shown in the response below:



The Status values are defined in the table below:

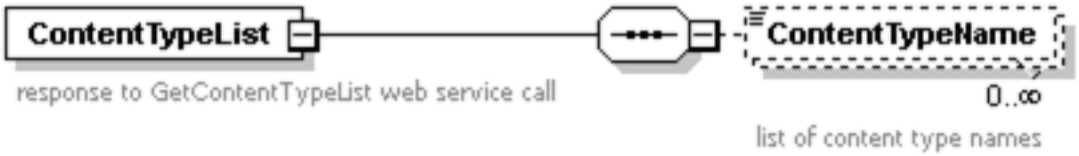
Status	Description
SUBMITTED	The message has been saved in the Message Connector storage but is not imported into Kofax Capture.
SUCCESS	The message has been successfully imported into Kofax Capture.
ERROR	The message could not be imported into Kofax Capture.
UNKNOWN	The message with the specified ID could not be found in the Message Connector storage. This can be caused by one of the following reasons: • A bad ImportID was specified in the TrackImport call • The message was processed (positive or negative) but it has been removed from the storage because the disk-space was required for new messages.

The ErrorString holds a detailed error description in case that import into Kofax Capture failed. The DeliveryTime is a dateTime field specifying when the import into Kofax Capture took place or failed or will be attempted, depending on the Status value. The Extension values are reserved for future extensions.

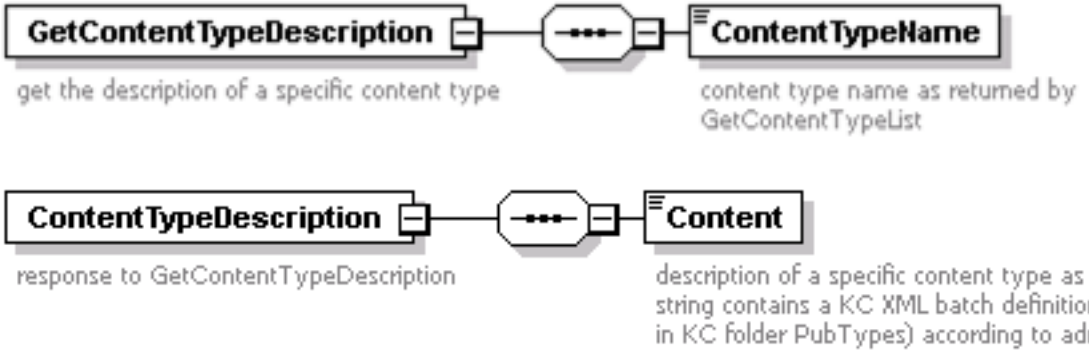
GetContentTypeList and GetContentTypeDescription Functions

These functions may be used by clients that need details (e.g. document classes, document fields, etc) about the published batch classes.

Function GetContentTypeList is used to get a list of all available configurations (e.g. batch classes with Kofax Capture). It does not require any input parameters. The response lists the name of each configuration.



Function GetContentTypeDescription is used to get the content of a specific configuration. It requires the configuration name returned by GetContentTypeList as input and returns the configuration as string. This string contains a KC XML batch definition (located in Kofax Capture folder PubTypes) according to admin.dtd.



Request	Response

`<GetContentTypeList/>`	`<ContentTypeList>`
	`<ContentTypeName>Order Forms`
	`</ContentTypeName>`
	`<ContentTypeName>batch_class2`
	`</ContentTypeName>`
	`</ContentTypeList>`

`<GetContentTypeDescription>`
`<ContentTypeName>Order Forms`
`</ContentTypeName>`
`</GetContentTypeDescription>`
	`<ContentTypeDescription>`
	`<Content>`
	`<?xml version="1.0" encoding="UTF-8"?>`
	`<!DOCTYPE AscentCaptureSetup SYSTEM ".. \Admin.dtd">`
	`<AscentCaptureSetup DatabaseVersion="29" ... >`
	`...`
	`</Content>`
	`</ContentTypeDescription>`

Example: Input of Unstructured Message

This example shows a request which inputs plain text and a TIFF file.

```
<Import>
   <Part>
     <ContentType>text/plain</ContentType>
     <Content>
      <Text>This is a sample text
line 2
line 3
end of text</Text>
     </Content>
   </Part>
   <Part>
     <Filename>page001.tif</Filename>
     <Content>
       <Binary>
         <!-- base64 encoded content of page001.tif -->
       </Binary>
     </Content>
   </Part>
</Import>
```

Example: Input of Customer-Specific XML Document

This example shows a sample request which inputs a customer XML and a Word document.

```
<Import>
   <StructuredXmlDocument>
      <ContentType>text/xml</ContentType>
      <Content>
        <Structure>
          <CustomType>
            <OrderNumber>12345</OrderNumber>
            <CustomerId>3434</CustomerId>
            <Items>
              <Item pcs=”1” article=”9999”/>
              <Item pcs=”5” article=”9124”/>
              <Item pcs=”3” article=”1299”/>
            </Items>
          </CustomType>
        </Structure>
     </Content>
   </StructuredXmlDocument>
   <Part>
     <Filename>document.doc</Filename>
     <Content>
       <Binary>
         <!-- base64 encoded content of document.doc -->
       </Binary>
     </Content>
   </Part>
</Import>
```

The customer XML can be rendered to PDF/TIFF or mapped to Kofax Capture fields.

Example: Compatibility with KCIC Web Services

The web service input of Kofax Import Connector does NOT provide a compatible web service interface, but it supports the use case to create a batch in Kofax Capture according to an XML document (e.g. according to ACEIDEF.dtd) and additional files (e.g. images). Kofax Import Connector supports this use case with its import function:

```
<Import>
   <StructuredXmlDocument>
      <ContentType>text/xml</ContentType>
      <Content>
        <Structure>
          <ImportSession>
            <Batches>
              <Batch name="batch_name">
                <Pages>
                  <Page ImportFileName="page001.tif”/>
                </Pages>
              </Batch>
            </Batches>
          </ImportSession>
        </Structure>
     </Content>
   </StructuredXmlDocument>
   <Part>
     <ContentType>image/tif</ContentType>
     <Filename>page001.tif</Filename>
     <Content>
       <Binary>
         <!-- base64-encoded content of page001.tif -->
       </Binary>
     </Content>
   </Part>
</Import>
```

The first part contains a batch XML element according to ACEIDEF.dtd. The second part contains an image which is linked via name “page001.tif”.

Chapter 2

XML Mapping

In this chapter we are using the sample files installed along with KC Plug-In to clarify the use of XML mapping. You can find the sample files in the Samples\DemoMapping directory of the installation ISO.

Consider the traditional Kofax order example of a company "Northwest Products". This company has been receiving order forms from their customers per fax for years. Now, however, they decided to save costs and intend to allow their customers to send structured XML order data instead of faxes.

Prerequisite: You can write the XSL transformation file manually or can generate it using any XML mapping visual tool, for example, Altova MapForce. Still, XSLT expertise is required to perform XML mapping.

In this chapter we provide examples how to achieve this goal by the means of simple and generic XML mapping. For the sake of simplicity, we chose only a subset of order data:

Northwest Products
525 Corporate Dr.
Lakeview, CA 90435

Ordered By:
Bill Slater
365 Planter Street
New York, NY
07326

67342 895

Ship To:

First Name: MARTIN
Last Name: JANEWAY

Address: 19 POWERS ROAD

City: MANHEIM
State: IL
Zip/Postal Code: 60420

Country: USA

Quantity	Item #	Description	Unit Price	Amount
1	638	LANG STERLING PIE SERVER	32.95	32.95

Subtotal	32.95
Sales Tax	2.55
Shipping	5.00
Total	40.50

Please check any items that apply:
☐ Do you wish to be on our mailing list? ☑ Do you wish to receive special offers?

Method of Payment:
☐ Check or Money Order Enclosed ☑ Purchase Order No. 42367

Please Bill: ☐ Visa ☐ MasterCard ☐ American Express

Credit Card Number

Expiration Date

X martin janeway
Authorized Signature

Each incoming order should create a new Kofax Capture batch of class KfxSampleXmlmappingBatch with one document with form type KfxSampleXmlMappingForm where:

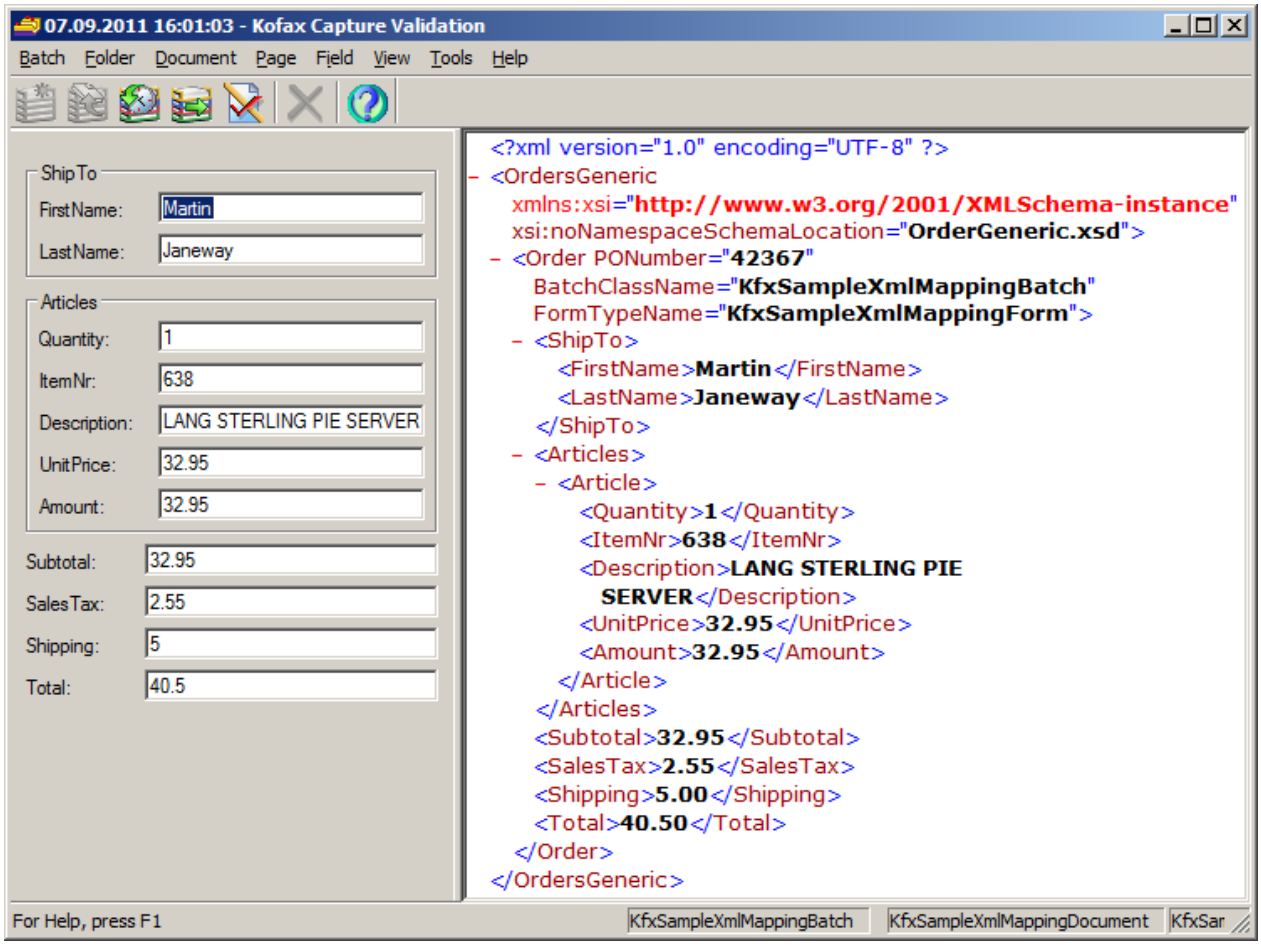
- Received XML data structure is always imported.
- Optionally, TIF images sent along with the structured XML data should be also imported.

The selected order data are to be mapped to the batch fields / index fields in the following way:

Input data fields	**Map to**		
	Batch fields	**Document tables**	**Document index fields**
Purchase Order No.	PONumber		
Ship To First Name		ShipTo	FirstName
Ship To Last Name			LastName
Quantity		Articles	Quantity
Item #			ItemNr
Description			Description
Unit Price			UnitPrice
Amount			Amount
Subtotal			Subtotal
Sales Tax			SalesTax
Shipping			Shipping
Total			Total

You can look at the KfxSampleXmlMappingBatch (also included in the samples folder) to see how these fields are defined in a batch class.

This is the desired outcome:



The following procedures provide an overview of configuration tasks needed to set up the mapping. Refer to configuration section of *Kofax Import Connector Administrators Guide* for a more detailed description of configuration steps.

Using Sample Files

There are four examples in the mapping subdirectories of Samples\DemoMapping directory named AutoXmlMapping, GenericMapping, SimpleMapping and SimpleMappingManual; and a matching Kofax Capture batch class in the file KfxSampleXmlMappingbatch.cab.

To install these samples on your test environment:

1. Import the KfxSampleXmlMappingBatch.cab class to your Kofax Capture and publish it.
2. Add the corresponding XML type to the KC Plug-In (XML type is comprised by the xsd schema and xml sample files in the corresponding mapping subdirectory - except for XML Import Connector compatible example, where the build-in XML type is being used).
3. Add a destination to KC Plug-In where the mapping should occur. Configure it correspondingly (don’t forget to set it batch class to “KfxSampleXmlMappingBatch”) and click “Show Files for

Visual Designer” in the ImportMapping Tab. Copy all files from the Samples\DemoMapping \<MappingType>XmlMapping\Sample<MappingType> to this directory (except for XML Import Connector compatible example, where the build-in XSL transformation is used).

4. As the simple input data you can use the provided sample XML file and the page002.tif file from the corresponding mapping subdirectory, but of course you can write your own “real” customer input XML data file (the provided trigger file can be used in the case of folder input driven by the trigger file).

Importing Structured Data via Simple XML Mapping

Northwest Products may select to use simple XML mapping. Their XML data could look like this:

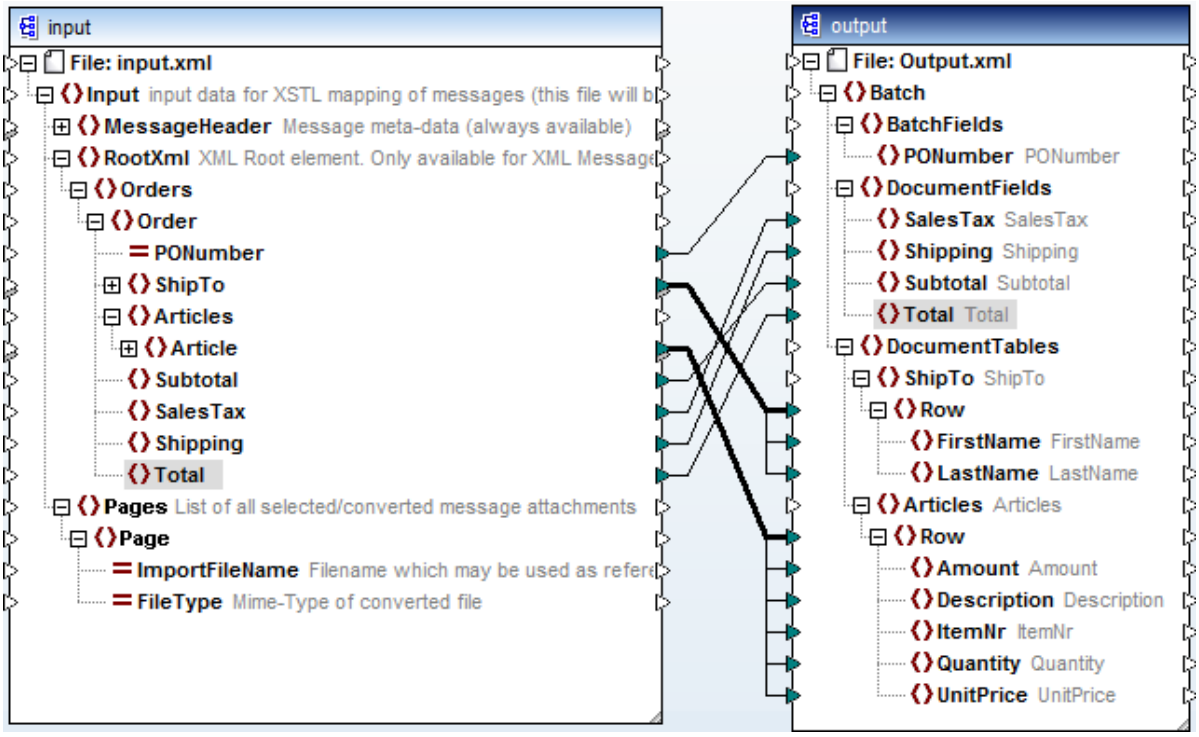
```
<?xml version="1.0" encoding="UTF-8"?>
<Orders xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="Order.xsd">
  <Order PONumber="42367">
    <ShipTo>
      <FirstName>Martin</FirstName>
      <LastName>Janeway</LastName>
    </ShipTo>
    <Articles>
      <Article>
        <Quantity>1</Quantity>
        <ItemNr>638</ItemNr>
        <Description>LANG STERLING PIE SERVER</Description>
        <UnitPrice>32.95</UnitPrice>
        <Amount>32.95</Amount>
      </Article>
    </Articles>
    <Subtotal>32.95</Subtotal>
    <SalesTax>2.55</SalesTax>
    <Shipping>5.00</Shipping>
    <Total>40.50</Total>
  </Order>
</Orders>
```

The schema file could look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Orders">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Order">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ShipTo">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="FirstName" type="xs:string"/>
                    <xs:element name="LastName" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="Articles">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Article">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="Quantity" type="xs:integer"/>
                          <xs:element name="ItemNr" type="xs:unsignedInt"/>
```

```
                        <xs:element name="Description" type="xs:string"/>
                        <xs:element name="UnitPrice" type="xs:decimal"/>
                        <xs:element name="Amount" type="xs:decimal"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="Subtotal" type="xs:decimal"/>
            <xs:element name="SalesTax" type="xs:decimal"/>
            <xs:element name="Shipping" type="xs:decimal"/>
            <xs:element name="Total" type="xs:decimal"/>
          </xs:sequence>
          <xs:attribute name="PONumber" type="xs:int" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

1. Add a new XML type "Orders" using the XML file and the schema file.
2. Add a new destination, assign it to the batch class KfxSampleXmlmappingBatch, document class KfxSampleXmlMappingDocument and form type KfxSampleXmlMappingForm. Do not forget to add a proper routing rule.
3. Select "Orders" as the XML type of the destination and select simple XML mapping.
4. Select to import original content (to Kofax Capture) but deactivate any conversions to TIF or PDF.
5. Create the desired mapping via MapForce, save it, and restart KC Plug-In.



Alternatively, if you do not have MapForce, you can use the sample files from Samples \DemoMapping\SimpleMapping\SampleSimpleMapping. Click **Show files for Visual Designer** and copy all sample files to this folder.

The customers of Northwest Products can now send their orders in XML format via email or file interface. The structure of the XML must match the defined type Orders. Optionally, they can attach a TIF image of the order. The Kofax Capture batch fields are populated automatically, the XML file and the (optional) TIFF image are imported as the document's pages.

Note With simple mapping:

- All attachments received along with the XML files are automatically imported to Kofax Capture to the same batch/document class as pages, without being mentioned in the XSL transformation.
- If the fields in the batch class where XML data elements are being mapped to have the same names as the XML data elements, the mapping is easier.

Importing Structured Data via XML Import Connector-Compatible Mapping

For some reasons Northwest Products may have decided to model their input data using the standard Kofax Capture XML Import Connector format. In order to do so, the XML input data would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<ImportSession xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Batches>
    <Batch BatchClassName="KfxSampleXmlMappingBatch">
      <BatchFields>
        <BatchField Value="42367" Name="PONumber"/>
      </BatchFields>
      <Documents>
        <Document FormTypeName="KfxSampleXmlMappingForm">
          <IndexFields>
            <IndexField Value="32.95" Name="Subtotal"/>
            <IndexField Value="2.55" Name="SalesTax"/>
            <IndexField Value="5.00" Name="Shipping"/>
            <IndexField Value="40.50" Name="Total"/>
          </IndexFields>
          <Tables>
            <Table Name="Articles">
              <TableRows>
                <TableRow>
                  <IndexFields>
                    <IndexField Value="1" Name="Quantity"/>
                    <IndexField Value="638" Name="ItemNr"/>
                    <IndexField Value="LANG STERLING PIE SERVER" Name="Description"/>
                    <IndexField Value="32.95" Name="UnitPrice"/>
                    <IndexField Value="32.95" Name="Amount"/>
                  </IndexFields>
                </TableRow>
              </TableRows>
            </Table>
            <Table Name="ShipTo">
              <TableRows>
                <TableRow>
                  <IndexFields>
                    <IndexField Value="Martin" Name="FirstName"/>
                    <IndexField Value="Janeway" Name="LastName"/>
                  </IndexFields>
                </TableRow>
```

```
            </TableRows>
          </Table>
        </Tables>
        <Pages>
          <Page ImportFileName="page002.tif"/>
        </Pages>
      </Document>
    </Documents>
  </Batch>
 </Batches>
</ImportSession>
```

As this is the standard Kofax format, the customer does not need to provide the XML schema file.

1. Add a new destination, assign it to the batch class KfxSampleXmlmappingBatch, document class KfxSampleXmlMappingDocument and form type KfxSampleXmlMappingForm. This information is only a fallback for the case that batch class information in the XML is not correct. Do not forget to add a proper routing rule.
2. Select "ImportSession" as the XML type of the destination and select XML Import Connector compatible mapping.
3. Select to import original but deactivate any conversions to TIF or PDF.
4. Restart KC Plug-In.

When using this standard Kofax format, MapForce is not needed.

In this scenario, attachments received along with the XML file are only imported to Kofax Capture if they are linked in the XML file (e.g. <Page ImportFileName="page002.tif"/>). The controlling XML document is never imported.

Importing Structured Data via Generic Mapping

For some reasons Northwest Products may have decided to use their own XML data format. They don't want to convert it to the standard Kofax Capture XML Import Connector format. However, they don't want to be bound to the same batch class/document class names. Instead, they want to provide additional attributes in the incoming XML data to select a particular batch class and form type. This requires generic XML mapping.

To link other documents from the controlling XML, all need to be imported at the same time: e.g., in the case of folder import, use trigger files or subfolder processing.

These are their sample XML input document and XML schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<OrdersGeneric xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="OrderGeneric.xsd">
  <Order PONumber="42367" BatchClassName="KfxSampleXmlMappingBatch"
 FormTypeName="KfxSampleXmlMappingForm">
    <ShipTo>
      <FirstName>Martin</FirstName>
      <LastName>Janeway</LastName>
    </ShipTo>
    <Articles>
      <Article>
        <Quantity>1</Quantity>
        <ItemNr>638</ItemNr>
        <Description>LANG STERLING PIE SERVER</Description>
        <UnitPrice>32.95</UnitPrice>
```

```
        <Amount>32.95</Amount>
      </Article>
    </Articles>
    <Subtotal>32.95</Subtotal>
    <SalesTax>2.55</SalesTax>
    <Shipping>5.00</Shipping>
    <Total>40.50</Total>
  </Order>
</OrdersGeneric>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="OrdersGeneric">
    <xs:complexType><xs:sequence>
        <xs:element name="Order" maxOccurs="unbounded">
          <xs:complexType><xs:sequence>
              <xs:element name="ShipTo">
                <xs:complexType><xs:sequence>
                    <xs:element name="FirstName" type="xs:string"/>
                    <xs:element name="LastName" type="xs:string"/>
                </xs:sequence></xs:complexType>
              </xs:element>
              <xs:element name="Articles">
                <xs:complexType><xs:sequence>
                    <xs:element name="Article">
                      <xs:complexType><xs:sequence>
                          <xs:element name="Quantity" type="xs:integer"/>
                          <xs:element name="ItemNr" type="xs:unsignedInt"/>
                          <xs:element name="Description" type="xs:string"/>
                          <xs:element name="UnitPrice" type="xs:decimal"/>
                          <xs:element name="Amount" type="xs:decimal"/>
                        </xs:sequence></xs:complexType>
                    </xs:element>
                </xs:sequence></xs:complexType>
              </xs:element>
              <xs:element name="Subtotal" type="xs:decimal"/>
              <xs:element name="SalesTax" type="xs:decimal"/>
              <xs:element name="Shipping" type="xs:decimal"/>
              <xs:element name="Total" type="xs:decimal"/>
            </xs:sequence>
            <xs:attribute name="PONumber" type="xs:int" use="required"/>
            <xs:attribute name="BatchClassName" type="xs:string"/>
            <xs:attribute name="FormTypeName" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence></xs:complexType>
  </xs:element>
</xs:schema>
```

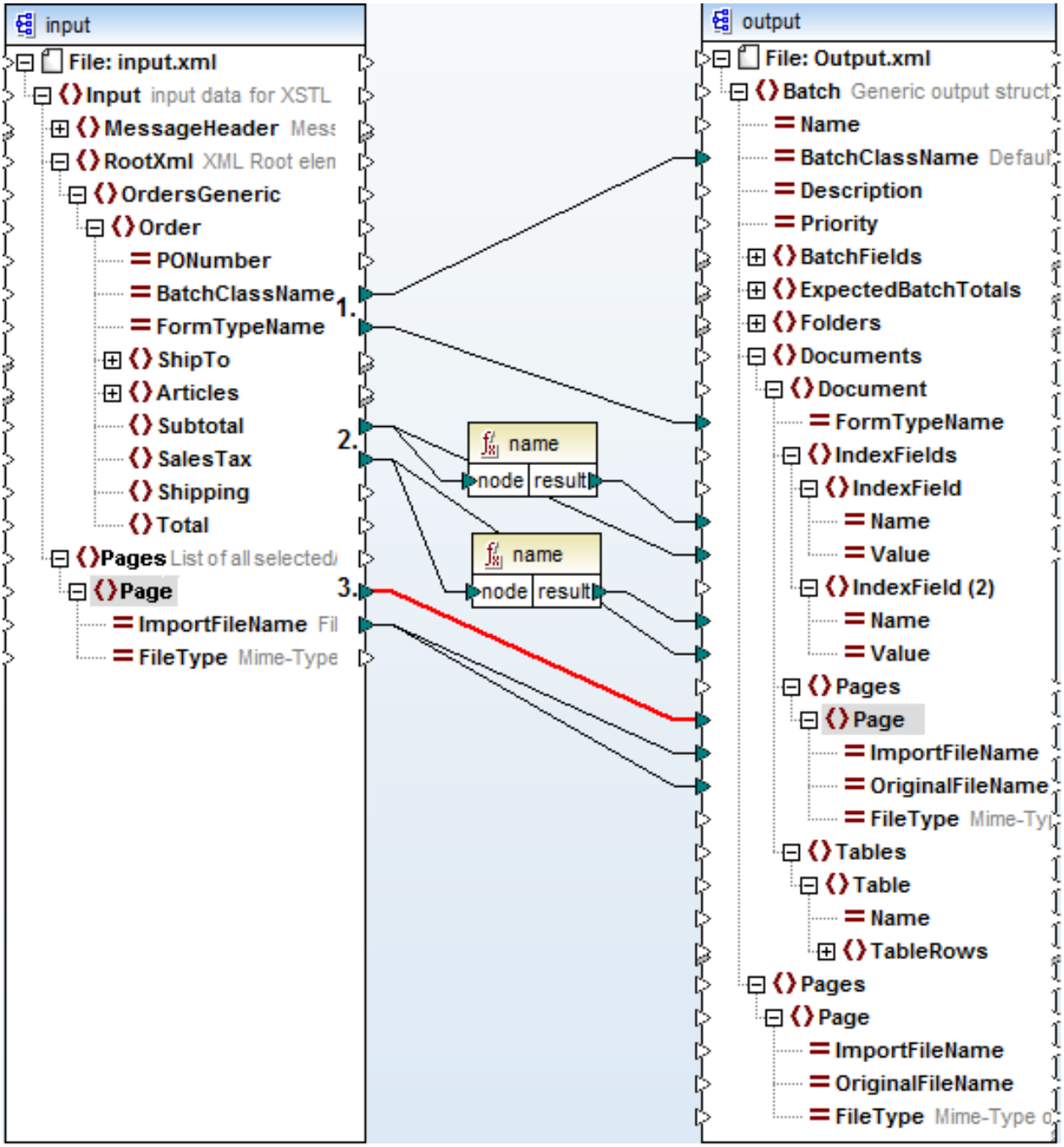
1. Add a new XML type "GenericOrders" using the XML file and the schema file.
2. Add a new destination. Assign it to any available batch class (this batch class will only be used if the batch class specified in the XML input is not available). Do not forget to add a proper routing rule.
3. Select "GenericOrders" as the XML type of the destination and select generic XML mapping.
4. Select to import original content (to Kofax Capture) but deactivate any conversions to TIF or PDF.
5. Create the desired mapping via MapForce and save it. Restart KC Plug-In afterwards.

   Alternatively, if you do not have MapForce, you can use the sample files from Samples \DemoMapping\GenericMapping\SampleGenericMapping. Click **Show files for Visual Designer** and copy all sample files to this folder.

   These are the usual steps in generic mapping:

   1. Map XML elements or attributes in the input.xml carrying class names directly to the corresponding attributes in the output.xml (see 1 in the screen shot below).

2. Required source elements or attributes in the input.xml document must be mapped to the corresponding name/value attribute pair in the output.xml document. The element's/attribute's name would control the name, and its value the value attribute in the corresponding position in the output.xml (see 2 in the screen shot below). This can be accomplished via MapForce's function block name(). If there are several element/attribute pairs to be mapped as separate index fields on the same output hierarchy (e.g. in the same document), it is necessary to duplicate the IndexField element in the output.xml to get more instances of the IndexField and map other source elements there. E.g., see how source elements "Subtotal" and "SalesTax" have been mapped in the step 2 below)
3. All received attachments and even the XML file itself are listed in the Pages sequence in the input.xml. If they also should be imported to Kofax Capture, the corresponding "ImportFileName" attribute must be explicitly mapped to the "ImportFileName" (and optionally to "OriginalFileName") attribute of the desired Page in the output.xml. See 3 in the screen shot.

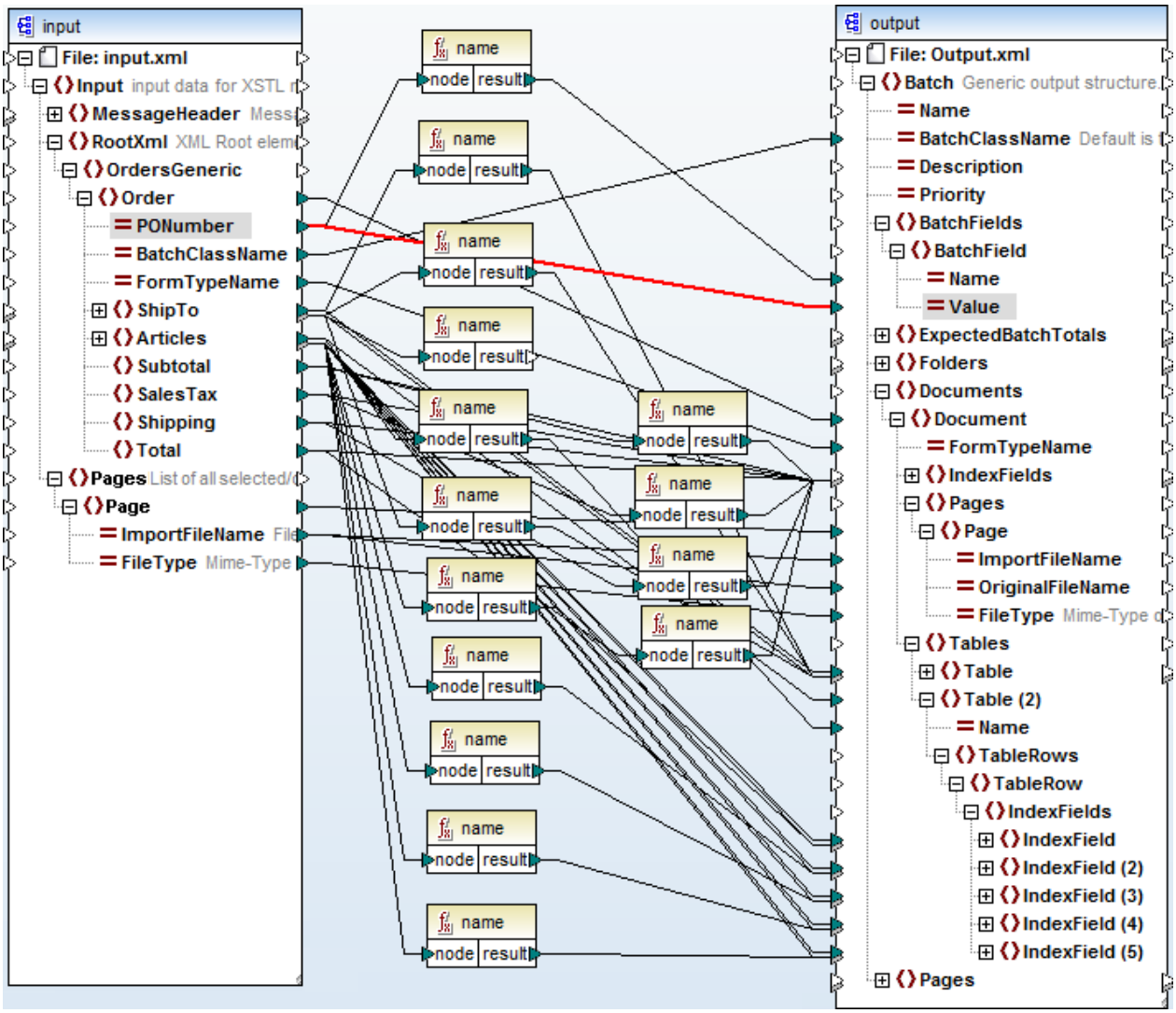


During the generation of the mapping in MapForce, it is possible to see how the sample XML document (of the destination's XML type) would be mapped using the current mapping. In our case it would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<Batch xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:/
ProgramData/Kofax/KCIC-E~1/KCPLUG~1/config/Schemas/SampleGenericDestination2/output.xsd"
 BatchClassName="KfxSampleXmlMappingBatch">
```

```
  <Documents>
    <Document FormTypeName="KfxSampleXmlMappingForm">
      <IndexFields>
        <IndexField Name="Subtotal" Value="32.95"/>
        <IndexField Name="SalesTax" Value="2.55"/>
      </IndexFields>
      <Pages>
        <Page ImportFileName="filename.ext" OriginalFileName="filename.ext"/>
      </Pages>
    </Document>
  </Documents>
</Batch>
```

Once all desired mappings are done, the final mapping would look like this:



Creating XSL Transformations Manually

Northwest Products can decide to model their data exactly in the same way as explained in the first simple mapping example, but they don't want to use the third party tool MapForce and asked their XSLT expert to create the XSLT file for them manually.

Proceed exactly in the same way as in the first example, and once your destination with simple XML mapping has been created, click **Show Files for Visual Designer**. The following files are created:

File Name	Purpose
Input.Xml	This is the sample input XML for the XSL transformation and consists of: • The message metadata in the MessageHeader element • Customer's input XML itself (the Orders element) • The list of all available attachments (the Files element) A similar file is internally generated during operation of Kofax Import Connector for each received customer XML document (see screen shot below)
Input.Xsd	This is the schema describing the input.xml structure. Both input XML and its schema files are always the same for both XML mapping variants – the simple and generic as well.
Output.Xml	This file defines the XML root element for the XML output of the XSL transformation.
Output.xsd	This is the schema describing the structure of output.xml. There is a substantial difference between output.xsd generated for simple and generic mappings: • With simple mapping, output.xsd is generated according to batch / folder / document fields defined in the batch class assigned to the destination where the mapping occurs. Therefore, if anything changes in the batch class definition in Kofax Capture, a different schema file is generated. Such a schema consists of four optional parts: • The sequence of all *BatchFields* (if any batch fields defined) • The sequence of all *DocumentTables* (if any document tables defined) • The sequence of all (non-table) *DocumentFields* (if any index fields defined) • The sequence of all *ExpectedTotals* (if any total index fields defined) • With generic mapping, output.xsd is always the same as it is not related to any specific batch class.

The Input.Xml could look e.g. like this:

```
<?xml version="1.0" encoding="utf-8"?>
<Input xsi:noNamespaceSchemaLocation="Input.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <MessageHeader>
    <KfxMessageCorrelation>1</KfxMessageCorrelation>
    <KfxMessageDeliveryPriority>0</KfxMessageDeliveryPriority>
```

```
    <KfxMessageDeliverySuspectedDupli>false</KfxMessageDeliverySuspectedDupli>
    <KfxMessageDeliveryType>TO</KfxMessageDeliveryType>
    <KfxMessageID>message_id_sample</KfxMessageID>
    <KfxMessagePages>1</KfxMessagePages>
    <KfxMessageReceptionTimeCreated>2001-12-17T09:30:47Z</KfxMessageReceptionTimeCreated>
    <KfxMessageSubject>message subject sample</KfxMessageSubject>
    <KfxOriginatorName>originator name sample</KfxOriginatorName>
    <KfxOriginatorNumber>originator@localhost</KfxOriginatorNumber>
    <KfxOriginatorService>EMAIL</KfxOriginatorService>
    <KfxRecipientName>Orders sample</KfxRecipientName>
    <KfxRecipientNumber>orders@localhost</KfxRecipientNumber>
    <KfxRecipientService>EMAIL</KfxRecipientService>
  </MessageHeader>
  <RootXml>
    <Orders xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <Order PONumber="42367">
        <ShipTo>
          <FirstName>Martin</FirstName> <LastName>Janeway</LastName>
        </ShipTo>
        <Articles>
          <Article>
            <Quantity>1</Quantity> <ItemNr>638</ItemNr>
            <Description>LANG STERLING PIE SERVER</Description>
            <UnitPrice>32.95</UnitPrice> <Amount>32.95</Amount>
          </Article>
        </Articles>
        <Subtotal>32.95</Subtotal>
        <SalesTax>2.55</SalesTax>
        <Shipping>5.00</Shipping>
        <Total>40.50</Total>
      </Order>
    </Orders>
  </RootXml>
  <Files>
    <File ImportFileName="filename.ext" />
  </Files>
</Input>
```

The XSL transformation in Kofax Import Connector involves the following actions:

1. Input.Xml structure is generated for each received message.
2. The XSL transformation is executed and its output is an internal representation of Output.Xml.
3. Output.Xml is parsed and corresponding batch fields are created/filled with the input data.

If you want to create XSL transformations manually, proceed as follows:

1. Consider your input.xml data (along with the schema file).
2. Consider the output data definition in output.xsd.
3. Create a XSL transformation that converts the input.xml like data to output.xml.
4. Save the created XSLT file as "XMLMapping.xslt" to the destination's visual designer folder. You can click **Show Files for Visual Designer** in the destination settings to open the folder.

If you don't want to create the file manually, you can use the sample file from the folder Samples \DemoMapping\SimpleMappingManual\SampleSimpleMappingManual.

An XSL transformation that fulfills the same task as in the simple XML mapping example could look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
```

```
  <xsl:template match="/">
    <Batch><xsl:apply-templates/></Batch>
  </xsl:template>
  <xsl:template match="MessageHeader"/>
  <!--Ignore MessageHeader-->
  <xsl:template match="Files"/>
  <!--Ignore Files-->
  <xsl:template match="RootXml/Orders/Order">
    <BatchFields>
      <PONumber><xsl:value-of select="@PONumber"/></PONumber>
    </BatchFields>
    <DocumentTables>
      <xsl:apply-templates select="ShipTo|Articles"/>
    </DocumentTables>
    <DocumentFields>
      <Subtotal><xsl:value-of select="Subtotal"/></Subtotal>
      <SalesTax><xsl:value-of select="SalesTax"/></SalesTax>
      <Shipping><xsl:value-of select="Shipping"/></Shipping>
      <Total><xsl:value-of select="Total"/></Total>
    </DocumentFields>
  </xsl:template>
  <xsl:template match="ShipTo">
    <ShipTo>
      <Row>
        <FirstName><xsl:value-of select="FirstName"/></FirstName>
        <LastName><xsl:value-of select="LastName"/></LastName>
      </Row>
    </ShipTo>
  </xsl:template>
  <xsl:template match="Articles">
    <Articles>
      <xsl:for-each select="Article">
        <Row>
          <Quantity><xsl:value-of select="Quantity"/></Quantity>
          <ItemNr><xsl:value-of select="ItemNr"/></ItemNr>
          <Amount><xsl:value-of select="Amount"/></Amount>
          <Description><xsl:value-of select="Description"/></Description>
          <UnitPrice><xsl:value-of select="UnitPrice"/></UnitPrice>
        </Row>
      </xsl:for-each>
    </Articles>
  </xsl:template>
</xsl:stylesheet>
```

Chapter 3

Web Services Sample Client

Kofax Import Connector includes two sample client applications for web service input. You can find them in CSharpClient.zip file on the installation ISO, in the Samples\WebService folder.

- Import.exe is a command line tool for importing documents to Message Connector.
- EDocGui.exe is a more advanced application with a graphical user interface.

For detailed information about the prerequisites, configuration, and operation of the sample clients, please refer to readme.txt (in the Export folder of CSharpClient.zip).

The source code of the applications are also included in the zip file, in the EDocGui and EDocImport folders. The sample programs are written .NET CSharp.

Chapter 4

Web Services Interface for Kofax Monitor

KC Plug-In and Message Connector offer web service interface functions for use with Kofax Monitor. The default URLs are

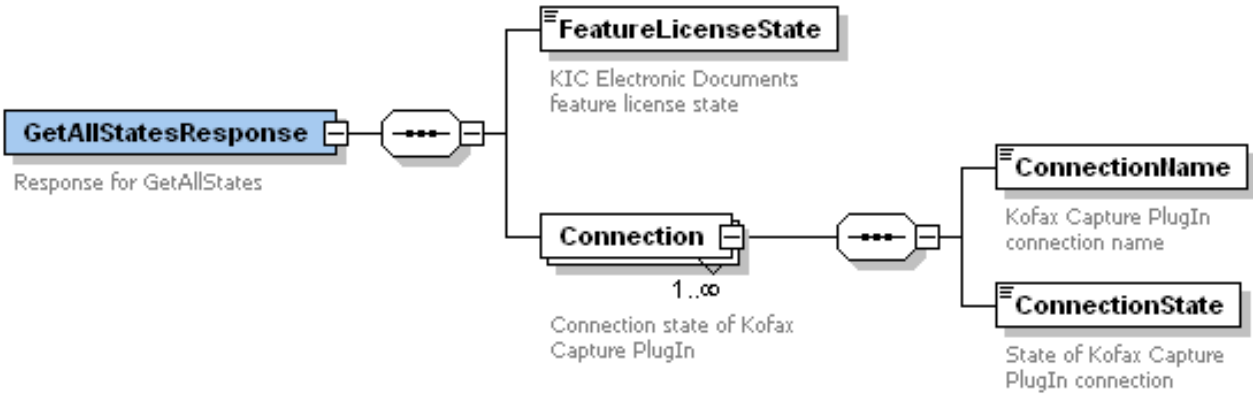
- https://<messageconnector>:25086/file/Monitor.wsdl
- http://<kcplugin>:8001/KIC-Electronic-Documents?wsdl

Replace <messageconnector> and <kcplugin> with the appropriate computer names. Additional information about using Kofax Monitor are available in the *Kofax Import Connector Administrator's Guide*.

KC Plug-In

GetAllStates Function

This function returns the status of Kofax Import Connector feature licenses and the status of all connections.



The FeatureLicenseState values are defined in the table below:

Value	Description
0	License OK
1	License invalid.
65536	Evaluation license found.

The ConnectionState values are defined in the table below:

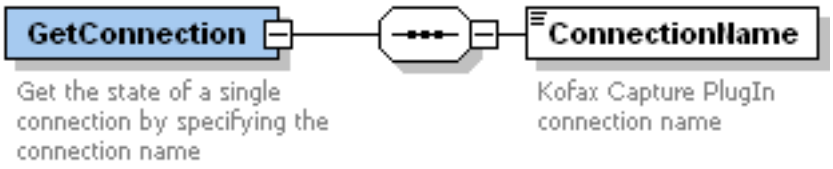
Value	Description
-1	Connection is active but not connected.
0	Connection is inactive.
1	Connection is active and connected.

Example:

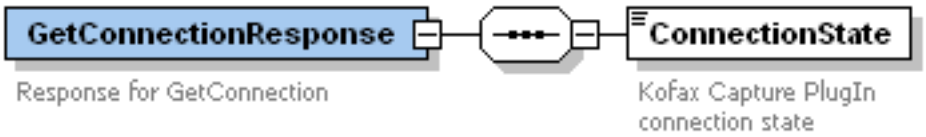
```
<s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
  <GetAllStatesResponse xmlns="http://www.kofax.com/2011/KIC-ElectronicDocuments">
    <FeatureLicenseState>1</FeatureLicenseState>
    <Connection>
      <ConnectionName>Connection2</ConnectionName>
      <ConnectionState>1</ConnectionState>
    </Connection>
    <Connection>
      <ConnectionName>Connection3</ConnectionName>
      <ConnectionState>0</ConnectionState>
    </Connection>
  </GetAllStatesResponse>
</s:Body>
```

GetConnection Function

This function returns the status of a connection between KC Plug-In and Message Connector.



On success, the function returns the status of the specified connection.



The ConnectionState values are defined in the table below:

Value	Description
-1	Connection is active but not connected.
0	Connection is inactive.
1	Connection is active and connected.

Example:

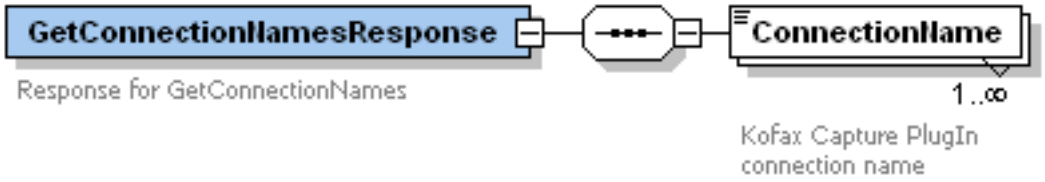
```
<s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
  <GetConnectionResponse xmlns="http://www.kofax.com/2011/KIC-ElectronicDocuments">
    <ConnectionState>-1</ConnectionState>
```

```
  </GetConnectionResponse>
</s:Body>
```

GetConnectionNames Function

This function returns the names of all connections between KC Plug-In and Message Connector.

On success, the function returns a GetConnectionNamesResponse containing the names of all connections.

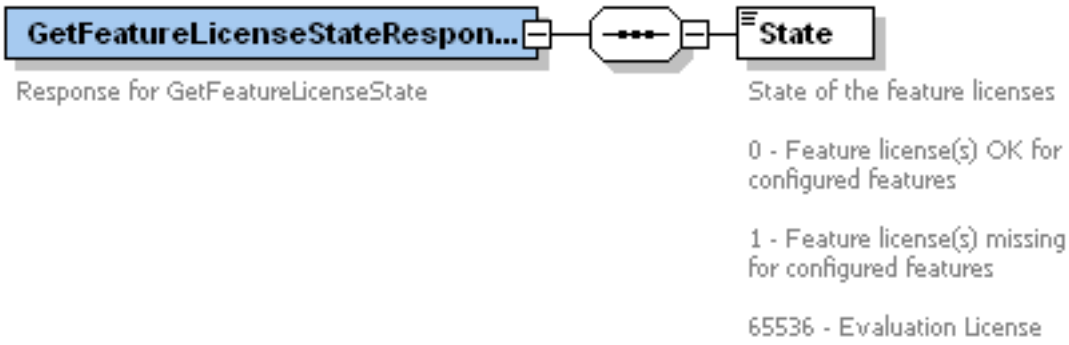


Example:

```
<s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
  <GetConnectionNamesResponse xmlns="http://www.kofax.com/2011/KIC-ElectronicDocuments">
    <ConnectionName>Connection2</ConnectionName>
    <ConnectionName>Connection3</ConnectionName>
  </GetConnectionNamesResponse>
</s:Body>
```

GetFeatureLicenseState Function

This function returns the status of Kofax Import Connector feature licenses.

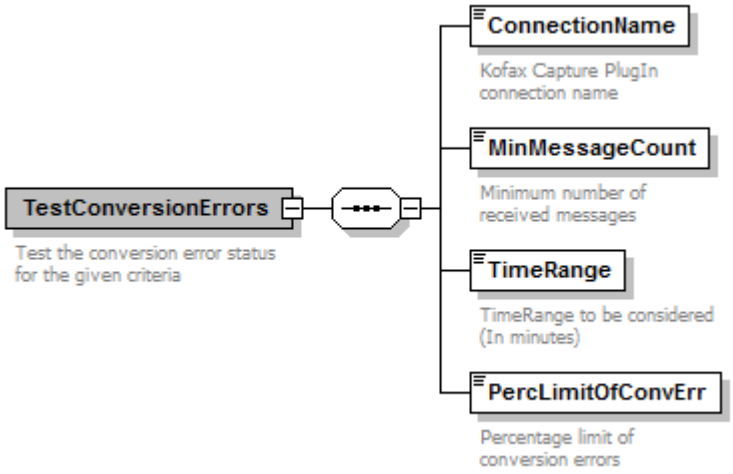


Example:

```
<s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
  <GetFeatureLicenseStateResponse xmlns="http://www.kofax.com/2011/KIC-ElectronicDocuments">
     <State>1</State>
  </GetFeatureLicenseStateResponse>
</s:Body>
```

TestConversionErrors Function

This function checks for the conversion errors in messages based on input criteria. The input schema is shown below:

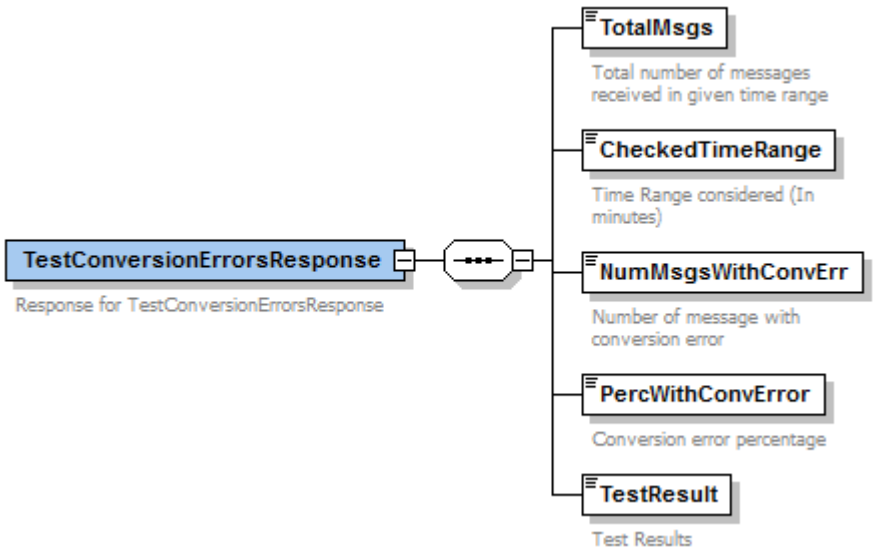


The input request fields are defined in the table below:

Request Fields	Description
ConnectionName	The name of the connection. This field is case-sensitive. Default value is "all".
MinMessageCount	Minimum number of messages to be received by the connection.
TimeRange (in minutes)	The time range for performing the test. Default value is 1440 minutes. Maximum value is 1440 minutes.
PercLimitOfConvErr	Minimum percentage of messages with conversion error for the test to be successful.

Note Restarting KC-Plugin service will set the message count to 0.

Based on the input criteria, the response is returned. The response schema is shown below



The output response fields are defined in the table below:

Response Fields	Description
TotalMsgs	The number of messages received during the specified time range.
CheckedTimeRange	Verified time range. This is same as the time range specified in the request.
NumMsgsWithConvErr	The number of messages with conversion error.
PercWithConvError	The percentage of messages with conversion error.
TestResult	This is the test result. Possible values are Success, Fail or NotEnoughMsgs.

Example: Sample Request

Input Request: For the test to be successful, at least 20 messages must be received by "Connection1" and the conversion error should be more than 30% in last 150 minutes.

```
<kic:TestConversionErrors>
   <kic:ConnectionName>Connection1</kic:ConnectionName>
   <kic:MinMessageCount>20</kic:MinMessageCount>
   <kic:TimeRange>150</kic:TimeRange>
   <kic:PercLimitOfConvErr>30</kic:PercLimitOfConvErr>
</kic:TestConversionErrors>
```

Output Response: As per the input criteria, "Connection1" received 20 messages in last 150 minutes and there are five document conversion errors, that is, 25%. Therefore, the criteria user tested is not achieved and the test result is fail.

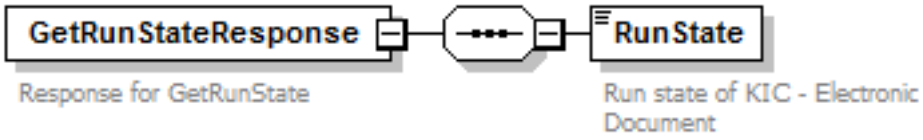
```
<TestConversionErrorsResponse xmlns="http://www.kofax.com/2011/KIC-ElectronicDocuments">
    <TotalMsgs>20</TotalMsgs>
    <CheckedTimeRange>150</CheckedTimeRange>
    <NumMsgsWithConvErr>5</NumMsgsWithConvErr>
    <PercWithConvError>25</PercWithConvError>
    <TestResult>Fail</TestResult>
```

```
</TestConversionErrorsResponse>
```

Message Connector

GetRunState Function

This function is used in conjunction with Kofax Monitor. It returns the run state of Message Connector. The function has no parameters and returns a response as shown below:



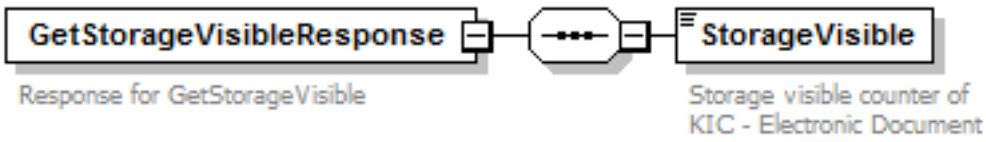
The RunState values are defined in the table below:

Value	Description
0	Not running
80	Storage full, no documents accepted into storage; import to Kofax Capture continues
100	Running

GetStorageVisible Function

This function is used in conjunction with Kofax Monitor. It returns how full is the storage of Message Connector, in percent.

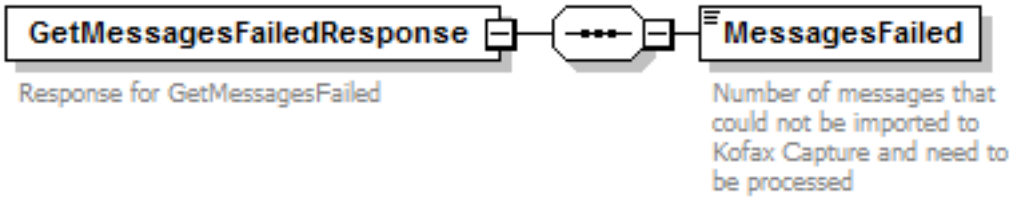
The GetStorageVisible function has no parameters and returns a response as shown below:



GetMessagesFailed Function

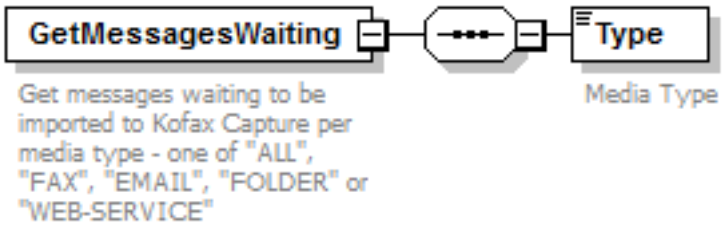
This function is used in conjunction with Kofax Monitor. It returns number of messages that could not be imported to Kofax Capture and need to be processed.

The GetMessagesFailed function has no parameters and returns a response as shown below:



GetMessagesWaiting Function

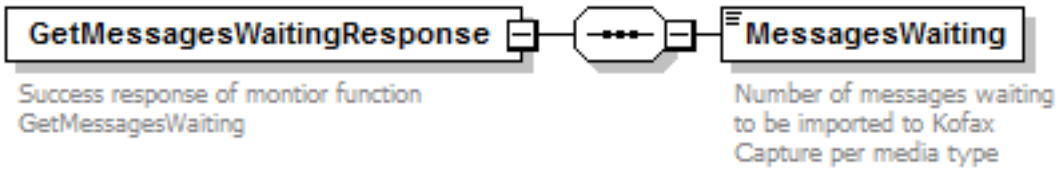
This function is used in conjunction with Kofax Monitor. It returns the number of messages waiting to be imported to Kofax Capture per media type.



The text parameter MediaType may have to following values:

- ALL (default)
- FAX
- EMAIL
- FOLDER
- WEB-SERVICE

The GetMessagesWaiting function returns a response as shown below:



Chapter 5

Scripting Interface

The KC Plug-In module contains a custom scripting interface that allows customizing how messages are imported into Kofax Capture. This is done via C# scripts. When configured, scripts run just before the content is imported into Kofax Capture and can modify or add field values or alter the binary content of the files, discard or add new binary content, or do any other alteration to the message fields or content. Additionally, scripts can e.g. make a call to a database to validate or look up data and then add looked up values to document or batch fields, or if desired abort message import and inform the Message Connector that a message has been rejected.

The KC Plug-In module can use

- scripts to create specific batch names
- custom document scripts to perform specific actions on the message fields or content before the message is imported into Kofax Capture
- scripts to reroute portion(s) of the document to a different destination

In addition to the KC Plug-In, a scripting DLL and a sample Visual Studio 2008 project are provided. These enable you to create, run, and test your scripts in Visual Studio before you deploy them to the KC Plug-In. Please note that the provided DLL is only for making debugging easier, neither the DLL nor Visual Studio are required to create scripts. Scripts do not have to be compiled before deployment. You can also create scripts in any text editor.

The sample project KCSImportScriptingSample.zip is located in `<programs>\Kofax\KIC-ED\KCPlugIn\ScriptSample\`.

The scripts, though written in C#, do not need to be compiled. You just have to configure the path to your script source file in the KC Plug-In configuration. When the plug-in starts, it will build the script on the fly and run it before your messages are imported or to get a batch name just before the batch is created.

The batch naming script has to implement a special *IBatchNameFormatter* interface. If an error occurs in the script, the connector writes the error to a log file and the default batch naming configured in the batch class is used. If the script is not configured the default batch name configured in the Kofax Capture batch class is used.

The custom document script has to implement a special *IDocumentScript2* interface. It runs just before a message is imported into Kofax Capture. In the script, you can make any desired changes to the document content.

The rerouting script has to implement a special *IBeforeMappingScript* interface. It runs after a document is retrieved from Message Connector but before field mapping. The script has full access to the document and it can send portions of the document back to Message Connector for later retrieval.

Tip To debug scripts using Visual Studio 2010, edit the script and add or uncomment the line

```
System.Diagnostics.Debugger.Break();
```

Additionally, open the script in Visual Studio 2010 and attach the script to the process Kofax.Kcs:KcImport.exe.

Adding References to Scripts

To add a reference to an assembly that is already in path, add a line beginning with //GAC: followed by the DLL names delimited by comma as a first line to your script.

```
//GAC:System.Data.dll
```

To add references to any other assemblies, add `//GAC:System.Data.dll` as the first line, and then add a line beginning with //ref: followed by the full paths to your DLLs delimited by comma as a second line to your script.

```
//GAC:System.Data.dll
```

```
//ref:c:\temp\myref1.dll,c:\temp 2\myref2.dll, c:\temp 3\myref3.dll
```

IBatchNameFormatter Interface Definition

This is the definition of the IBatchNameFormatter interface

```
using System;
using System.Collections.Generic;
using System.Text;
using Kofax.KCS.ImportConnector.Messages;

namespace Kofax.KCS.ImportConnector.Scripting
{
 public interface IBatchNameFormatter
 {
 string GetBatchName(ReadonlyMessage[] msg);
 }
}
```

The ReadonlyMessage[] array contains all the messages that are part of a batch. If it is a single message batch, then the array contains only 1 message at position 0. All the properties and fields values are read-only. You cannot change any of those values using this interface. You can use any of the defined Kofax Capture variables, e.g. "{Sequence Number}". Kofax Capture variables are translated to Kofax Capture values. (Please refer to the Kofax Capture documentation for more information about Kofax Capture variables).

The returning string from the function is the actual batch name.

Note If you write custom batch naming scripts you should pay attention that the batch names must be unique, otherwise the batch creation in Kofax Capture will fail.

ReadonlyMessage properties

Property Name	Type	Description
Fields	IDictionary <string, string>	The collection of Key-Value - pairs containing field names in the key values and the field values in the value values. The collection contains the standard message fields listed below and any extension fields if they are defined in Message Connector. (Message fields are information that Message Connector delivers with each message. Message extension fields are currently not provided by Message Connector and should not be used. They are reserved for future use.) Standard message fields are described in the Administrator's Guide
DestConfig	Object (Destination Config)	The configuration of the destination to which the message has been redirected. Do NOT modify the properties of this object!

IDocumentScript2 Interface Definition

This is the definition of the IDocumentScript2 interface

```
public interface IDocumentScript2
 {

/// <summary>
 /// Called before the import content and import order is determined
 /// Body and attachment content and import order can be manipulated in this function
 /// </summary>
 /// <param name="messageBody">A read-only instance of the message being imported</param>
 /// <param name="messageBody">The list of the received message bodies.</param>
 /// <param name="attachments">The list of the received attachments.</param>
 /// <param name="extension">Reserved for future use.</param>
 void ManageMessageFiles(ReadonlyMessage message,
 List<Attachment> messageBody,
 List<Attachment> attachments,
 object extension);
 /// <summary>
 /// Called before a document is imported into KC
 /// </summary>
 /// <param name="indexFields">The list of index fields defined in the configured document
 class.
 /// If document class not defined this will be empty.</param>
 /// <param name="folderFields">The list of folder fields defined in the configured folder
 class.
 /// If folder class not defined this will be empty.</param>
 /// <param name="batchFields">The list of batch fields defined in the configured batch
 class.</param>
 /// <param name="messageBody">The list of the received message bodies.</param>
 /// <param name="attachments">The list of the received attachments.</param>
 /// <param name="extension">Reserved for future use.</param>
 void BeforeDocumentImport(IDictionary<string, string> indexFields,
 IDictionary<string, string> folderFields,
 IDictionary<string, string> batchFields,
 List<Attachment> messageBody,
```

```
List<Attachment> attachments,
object extension);

/// <summary>
/// Called before a message is imported into KC
/// </summary>
/// <param name="indexFields">The list of index fields defined in the configured document
class.
/// If document class not defined this will be empty.</param>
/// <param name="folderFields">The list of folder fields defined in the configured folder
class.
/// If folder class not defined this will be empty.</param>
/// <param name="batchFields">The list of batch fields defined in the configured batch
class.</param>
/// <param name="messageBody">The list of the received message bodies.</param>
/// <param name="attachments">The list of the received attachments.</param>
/// <param name="extension">Reserved for future use.</param>
void BeforeMessageImport(IDictionary<string, string> indexFields,
IDictionary<string, string> folderFields,
IDictionary<string, string> batchFields,
List<Attachment> messageBody,
List<Attachment> attachments,
object extension);
}
```

If defined, the ManageMessageFiles function implementation is run before the import content and import order is determined. Here you can manipulate the import content: add new content, discard content, and change existing content. You can discard binary content by setting a DoImport flag of the binary content to false to skip the import of that attachment or body.

If defined, the BeforeDocumentImport function implementation runs before each Kofax Capture document is created.

If defined, the BeforeMessageImport function implementation runs before each message is imported into Kofax Capture

The input parameters for the BeforeDocumentImport and BeforeMessageImport methods are the list of all folder, document and batch field values, the list of all message bodies, and the list of all attachments. Here you can manipulate batch, folder, and document field values. The list of bodies and attachments are only for reading in these functions. You cannot change the import content. If you need to manipulate the import content this should be done in ManageMessageFiles.

The indexFields parameter contains a key-value pair list containing all the index fields defined for the used document class. If a document class is not configured or if there are no index fields defined, this will be empty. The key property contains the index field name and the value property contains the index field value.

The folderFields parameter contains a key-value pair list containing all the folder fields defined for the used folder class. If a folder class is not configured or if there are no folder fields defined, this will be empty. The key property contains the folder field name and the value property contains the folder field value.

The batchFields parameter, similar to the indexFields and folderFields parameter contains a key-value pair list containing all the batch fields defined for the used batch class. If there are no batch fields defined, it is empty. The key property contains the batch field name and the value property contains the batch field value.

The messageBody parameter contains all selected representations of the message body (original, PDF, TIF).

The attachments parameter contains all selected representations of the attachments (original, PDF, TIF).

Attachment class properties

Property Name	Type	Description
Extension	String	The extension of the received file
LongOrBinaryFileName	String	The file name of the received attachment/body. The connector first looks for the long file name, if it does not exist, then it takes the binary file name, which is the short file name.
HierarchicalPosition	String	MC hierarchical position of the file.
DoImport	Bool	By default it is true. If you set this value to false in your script, this attachment/body will not be imported into Kofax Capture
Content	Byte[]	The binary content of a body/attachment. If it is text, then it is encoded using the default Windows encoding for on the computer where the connector runs.
DocConversionError	String	If there was a document conversion error for this document, this property will contain the description of the error
IsOriginal	Bool	Indicates if a file is an original file or a converted file
IsImage	Bool	Indicates if it is an image or another file format
IsOriginalEml	Bool	Indicates if it is the EML representation of the original message
ContentID	String	MIME content ID. If not set by Message Connector the value is null.
ContentDisposition	String	MIME content disposition. If not set by Message Connector the value is null.
CreationDate	DateTime	MIME creation date of original attachment. If not set by Message Connector the value is 01.01.0001 00:00:00.
ModificationDate	DateTime	MIME modification date of the original attachment. If not set by Message Connector the value is 01.01.0001 00:00:00.
ContentType	String	MIME type of the original attachment. If not set by Message Connector, the KC Plug-In will set it depending on the file extension of the attachment.
ActualType	String	Type of the attachment set by Message Connector. Possible values: TEXT , BINARY_IMAGE, BINARY_nonIMAGE, ROOT_XML.
IsXmlRendering	Bool	Indicates that the attachment is the result of a converted XML document.
OriginalFileName	String	Contains the original file name of the attachment with the original extension (e.g., when a document has been converted, it now has a different name/extension).

Rejecting Messages from Script

You can perform checks and reject messages with scripts that implement the IDocumentScript2 interface. To reject a message, throw an exception of the type ScriptException with the desired description as an argument. The description is then displayed in Message Connector Monitor.

Ignoring Messages from Script

You can perform checks and ignore messages with scripts that implement the IDocumentScript2 interface. To ignore a message, throw an exception of the type ScriptIgnoreMessageException with the desired description as an argument. In order to work correctly ScriptIgnoreMessageException needs to be thrown from the function ManageMessageFiles. Optionally, when throwing the exception you can also select not to send a notification and not to archive the message by passing false as value for doNotifyArchive in the exception constructor. Similar to reject, ignore causes the message not to be imported into Kofax Capture. The difference is however, that ignore will send a positive confirmation back to Message Connector and provides the option to turn off archiving and notifications for such messages.

Using BeforeDocumentImport to Access the Current Attachment

You can use the BeforeDocumentImport function to access the current attachment using a script.

```
public void BeforeDocumentImport(IDictionary<string, string> indexFields,
    IDictionary<string, string> folderFields,
    IDictionary<string, string> batchFields,
    List<Attachment> messageBody,
    List<Attachment> attachments,
    object extension)
```

There are two lists which are accessible in the script: "messageBody" and "attachments". The "object extension" parameter is enhanced to store information about the current attachment. This allows you to select the relevant attachment from the parameter "messageBody" or "attachments"

Parameter	Description
CurrentMsgInfo.currentImport	Stores the information about type: attachment is of Body type or attachment type.
CurrentMsgInfo.currentAttachment	Stores the index of current attachment in the list.
CurrentMsgInfo.currentPageList	Stores the index of all the pages in the attachment list. "currentPageList" property should be used only when the destination is configured for "XMLImport Connector compatible" or "Generic" XML mapping. In all other cases, "currentAttachment" property should be used.

Procedure to know the current attachment:

1. Check if the current attachment is in the "attachments" list of "messageBody" list.
   Use the "extension" object:

```
Dictionary<string, object> arguments = (Dictionary<string, object>)extension;

CurrentMsgInfo info = (CurrentMsgInfo)arguments["CurrentMsgInfo"];
Attachment currAtt = null;

if (info.currentImport == ImportData.ATTACHMENT)
   //current Attachment is in attachments list
else if (info.currentImport == ImportData.MESSAGE_BODY)
   //current Attachment is in messageBody list
else if (info.currentImport == ImportData.PAGE_LIST)
  //current document has list of pages (attachments)
```

Note the following:

- `CurrentMsgInfo info.currentImport` will always return `ImportData.MESSAGE`, if all of the following conditions are true:
  - "Create document per attachment" is not selected, and
  - XML mapping is not configured for "XMLImport Connector compatible" or "Generic xml"
- When XML mapping in the destination is configured for "XMLImport Connector compatible" or "Generic xml":
  - Documents are created as per the XML data and it does not depend on "Create document per attachment" option.
  - `currentImport` property of `CurrentMsgInfo` will return `ImportData.PAGE_LIST`.

2. Find the index of the current attachment in the list.
   - Info.currentAttachment holds the index of current attachment.
   - If Info.currentAttachment is less than 0, the current Import is of type message.
   - If info.currentPagelist is not null, current document has list of pages (attachments.)
3. Access the current attachment using list and current index.

```
if (info.currentImport == ImportData.ATTACHMENT)
    currAtt = attachments[info.currentAttachment];
else if (info.currentImport == ImportData.MESSAGE_BODY)
    currAtt = messageBody[info.currentAttachment];
else if (info.currentImport == ImportData.PAGE_LIST)
    {
      currPageList = new List<Attachment>();
      for (int i = 0; i < info.currentPageList.Count; i++)
      currPageList.Add(attachments[info.currentPageList[i]]);
    }
```

Sample code that can be used in script file's BeforeDocumentImport() function:

```
public void BeforeDocumentImport(IDictionary<string, string> indexFields, IDictionary<string,
 string> folderFields, IDictionary<string, string> batchFields, List<Attachment> messageBody,
 List<Attachment> attachments, object extension)
    {
        Dictionary<string, object> arguments = (Dictionary<string, object>)extension;

        CurrentMsgInfo info = (CurrentMsgInfo)arguments["CurrentMsgInfo"];
        Attachment currAtt = null;

        if (info.currentImport == ImportData.ATTACHMENT)
            currAtt = attachments[info.currentAttachment];
        else if (info.currentImport == ImportData.MESSAGE_BODY)
            currAtt = messageBody[info.currentAttachment];
        else if (info.currentImport == ImportData.PAGE_LIST)
            {
                currPageList = new List<Attachment>();
                for (int i = 0; i < info.currentPageList.Count; i++)
                    currPageList.Add(attachments[info.currentPageList[i]]);
            }
    }
```

Class added in API:

```
public class ImportData
{
    public const string MESSAGE="MESSAGE";
    public const string MESSAGE_BODY="MESSAGE BODY";
```

```
    public const string ATTACHMENT="ATTACHMENT";
}
public class CurrentMsgInfo
{
    public string currentImport = "";
    public int currentAttachment = -1;
    public List<int> currentPageList=null;
}
```

Note Fix the line breaks if you copy and paste the code from this guide.

Identification of Message Type based on CurrentImport and CurrentAttachment

"Create document per attachment" UI Option	CurrentImport	Message Type	CurrentAttachment
Clear	`ImportData.MESSAGE`	Entire message	-1
Selected	`ImportData.MESSAGE_BODY`	Body	currentAttachment >=0 Attachment currAtt = messageBody[info.currentAttachment];
	`ImportData.ATTACHMENT`	Attachment	currentAttachment >=0 Attachment currAtt = attachments[info.currentAttachment];
Not applicable	`ImportData.PAGE_LIST`	Page list	-1 Current page list can be accessed using following code: `List<Attachment> currPageList = new List<Attachment>(); for (int i = 0; i < info.currentPageList.Count; i++) currPageList.Add(attachments[info.currentPageList[i]]);`

Using BeforeDocumentImport to get the EML/MSG/ZIP filename

You can use the BeforeDocumentImport function to get the EML/MSG/ZIP filename.

To use this script, do the following:

1. Extract KCSImportScriptingSample.zip (default: `C:\Program Files (x86)\Kofax\KIC-ED\KCPlugIn\ScriptSample`) to a local directory.
2. Open **Additional settings** tab in KC Plug-In **Destination Configuration**. For **Advanced batch/document script path** option, select **Browse** and locate the IDocument2_FR6371.cs file.
3. Select **Create document per attachment** in **Import mappings** tab.

4. In the Kofax Capture batch class configuration, create a document index field to get the MSG/EML/ZIP file name.
5. Restart the KC Plug-in service.

Note The default name of the field is KfxExtractedFromFile (case sensitive). If you want to use a different field name, modify the string constant INDEX_FIELD_EXTRACTED_FROM_FILE in the script and publish the batch class.

Additional Settings

To get the name of the email in the ExtractedFrom field, do the following:

- Configure the **Import trigger file** option in the **Advanced Folder Import Settings**.
- Configure the trigger file extension (for example, .trg) in **Skip importing files with formats** for destinations you do not want to import the trigger file into Kofax Capture.

IReRouteScript Interface Definition

This script is called after KC Plug-In receives a document from Message Connector, but before field mapping, XML mapping / rendering, or VRS processing. It has full access to the document and can modify it. The script can also insert or change the root XML and metadata fields used by rules.

Additionally, this script can split a document into multiple XML documents and return them to the Message Connector. The individual documents can be subsequently reimported to KC Plug-In.

```
public interface IReRouteScript
    {
        /// <summary>
        /// Called reroute script
        /// </summary>
        /// <param name="message">The complete message, can be modified.</param>
        /// <param name="extension">Reserved for future use</param>
        //   1 : continue (withoutRefetch)
        //   2 : continue with refetch
        //   3 : Reselect destination without refetch.
        //   4:  Reselect destination with refetch
        //   5:  split docs : for EDI
        //   6:  other
        eMessageScriptCode ReRoute(Message message, object extension);
    }
```

The script returns one of the following options which determine how to proceed with a document:

Return value	Description
Continue	The destination that is selected in script is used and cannot change. Rules are ignored. It assumes that the user called ReRoute function from script. No refetch is performed.
SplitDocs	This is used for EDI file when an EDI files contains multiple EDI messages. Usage can be checked in the file EDIReRouteScriptFile.cs from `C:\Program Files (x86)\Kofax\KIC-ED\KCPlugIn\Bin\script\`.

Return value	Description
ContinueWithRefetch	The destination that is selected in script is used and cannot change. Rules are ignored. It assumes that the user called ReRoute function from script. Any changes to the message from the script are discarded, the complete message is again fetched from the storage and it is processed according to the new destination.
ReSelectDestinationWORefetch	The message goes through the rules. If no rules match, messages go to the default destination. No refetch is performed. The import configuration of the selected destination (for example conversion options) is ignored. It is assumed that the script has done all necessary work and rules are just used to select the appropriate destination. If you want to import EDI documents as EML or MSG, you have to configure this in the routing destination. The default EDI script is using ReSelectDestinationWORefetch as return value. As a result, if importing as EML or MSG is enabled for the final destination, this setting is ignored.
ReSelectDestinationWithRefetch	Any changes to the message from the script are discarded, the complete message is again fetched from the storage. The message goes through the rules. If no rules match, messages go to the default destination. A message refetch is performed according to the new destination by calling ViewMessage.

"'IReRouteScript" is the main interface, however, in order to change the destination and reroute the message to other destination, implement a new script derived from the class Kofax.KCS.ImportConnector.Config.ReRoutingScript.

Implement the function `public abstract eMessageScriptCode ReRoute(Message message, object extension)`. In this function you can check conditions like message size, message extension, etc., to change the destination. ReRoute must return values as defined in the table above. Additionally, you can change the destination using the function `protected void ChangeDestination(Message message, string destinationName)`.

Sample usage of the ReRoutingScript class can be found in the file MsgReRouteScriptFile.cs located in `C:\Program Files (x86)\Kofax\KIC-ED\KCPlugIn\Bin\script\`.

Kofax Import Connector includes the script EDIReRouteScriptFile.cs, located in `C:\Program Files (x86)\Kofax\KIC-ED\KCPlugIn\Bin\script\`. This script is used for importing of EDI documents. You can modify this default script, however, your changes might be overwritten on software reinstall or update.

Chapter 6

Custom Conversion Script

The document conversion function in Message Connector knows many file types (extensions) and selects the appropriate document conversion tools and options automatically. The custom conversion script allows you to configure an additional list of file extensions that should be converted to PDF.

Configuring Custom Conversion Script

Message Connector must be configured to take advantage of the custom conversion script.

1. In the Message Connector Configuration, **Document Conversion** tab, in the **Custom Extension List** field, specify a blank separated list of file extensions (without dot) that your script is going to convert to PDF. Save the configuration and restart the Message Connector service.
2. Create a batch file as shown below that performs the conversion to PDF. Save is as “CustomToPdf.bat” to the Scripts subfolder of the Message Connector installation directory.
3. Test the conversion with the “Convert Document” test page of the Message Connector Web Portal

Sample Script

A simple template for the “CustomToPdf.bat” script is shown here; instead of converting to PDF the source file is only copied to the target.

```
@ECHO OFF
REM A simple custom conversion script example.
REM The source file is copied to the destination file.
REM A productive script has of course to create a PDF file from the source file.
REM For examples see the files of the Scripts folder.
REM Parameters:
REM %1 SourceFile
REM %2 TargetFile
REM %3 Utf8TextFile (0 - Other, 1 - Utf8)

ECHO Called: Custom2pdf.bat %*
setlocal

REM Copying source file to destination file
copy %1 %2

ECHO Error level=%errorlevel%
```

You can find examples that actually perform a conversion in the Scripts folder.

Chapter 7

Custom Storage Strings

When you configure a destination in KC Plug-In, you can map some of the document metadata to document and folder fields of a Kofax Capture batch class.

However, only a subset of document metadata is available for use with the user interface. All metadata is available in Kofax Capture as custom storage strings (for documents and folders).

Custom storage strings can be read via the Kofax Capture API and could be used for example in an export connector or in custom scripts.

Most custom storage strings are available for each document and for each folder. Refer to the Description column for exceptions.

Custom storage string	Description
ED_CSS_KfxOriginatorService	The service name of the originator. In case of fax FAX and in case of email EMAIL.
ED_CSS_KfxOriginatorNumber	The fax number of the originator (caller ID) or originator email (mime-header/from/mailbox/address) for POP3/ SMTP mail.
ED_CSS_KfxOriginatorName	The full name of the message originator. For faxes, this is the TSI. For SMTP and POP3, it's the originator display name (mime-header/from/mailbox/displayname).
ED_CSS_KfxRecipientService	The service name of the recipient. In case of fax FAX and in case of email EMAIL.
ED_CSS_KfxRecipientNumber	The recipient fax number (called party number) or email address (for SMTP it's the first active email recipient, for POP3 the mailbox user name).
ED_CSS_KfxRecipientName	The full name of the message recipient. For POP3, this is the mailbox display name.
ED_CSS_KfxMessageCorrelation	The correlation information of the message (for internal use).
ED_CSS_KfxMessageID	The unique ID of the message given by the Message Connector on message arrival.
ED_CSS_KfxMessageSubject	The subject of the message. For faxes, this is "Fax from" + TSI.
ED_CSS_KfxMessageReceptionTimeCreated	MAIL: The time when the Message Connector retrieved the message. FAX: The time when the fax server received the message.

Custom storage string	Description
ED_CSS_KfxMessageReceptionCallerId	The number of the sending fax machine. (Optionally available for FoIP, Biscom, KCS)
ED_CSS_KfxMessagePages	The number of fax pages in the message.
ED_CSS_KfxMessageDeliveryType	The type of delivery of the message. Reserved for future use. Currently static string "TO".
ED_CSS_KfxMessageDeliveryPriority	The priority of the message. Reserved for future use. Currently static value "1".
ED_CSS_KfxMessageDeliverySuspectedDupli	Reserved for future use. Currently static value "0".
ED_CSS_KfxMessageReceptionTimeReceived	The reception time of fax.
ED_CSS_KfxRoutingNumber	The meaning of this message field is different for various inputs • FAX: Extension (called-party-number) • SMTP: active email recipient • POP3/IMAP: mailbox user name • FOLDER: full path to the imported file • WEBSERVICE: empty string
ED_CSS_KfxMessageTimePosted	Only available for email messages, this field contains the date and time when the message was sent.
ED_CSS_KfxRecipientsTo	Comma separated list of "To" recipients.
ED_CSS_KfxRecipientsCc	Comma separated list of "Cc" recipients.
ED_CSS_KfxRecipientsBcc	Comma separated list of "Bcc" recipients.
ED_CSS_Batchname	The name of imported Kofax Capture batch.
ED_CSS_Batchclass	The batch class name of imported Kofax Capture batch.
ED_CSS_Import-Date-Short	The import date in the form YYYY-MM-DD.
ED_CSS_Import-Date-Long	The import date in the form YYYY-MM-DDTHH-MM-SS.
ED_CSS_KfxImportFolderName	Folder name of the sub folder from which messages are polled. Example, if the polling folder is "Inbox" and sub folder is "sub1", then the field will contain "Inbox/sub1".
ED_CSS_Batch-Id	The batch ID of the imported Kofax Capture batch.
ED_CSS_VRS-Error	The VRS error description in the case of VRS error. (Document only)

Custom storage string	Description
ED_CSS_OriginalFileName, ED_CSS_OriginalFileNamePageNr	For documents, this string stores the full path of the original content file. This string is available when you enable "Include original content" in the destination configuration and "Originals import mode" is set to "Save to disk". For folder, this string stores the full path of the saved EML file. This string is available when you enable "Include complete message as EML file" in the destination configuration and "Originals import mode" is set to "Save to disk". PageNr is the page number of the first page of the related converted document in Kofax Capture. E.g.: • ED_OriginalFilename1: C:\..\Images\00000CC4\200\1_original.tif • ED_OriginalFilename3: C:\..\Images\00000CC4\200\3_original.tif
ED_CSS_DocPerAttachment	The document creation mode used for import (Document only): • True: One document per each attachment • False: One document per message

Chapter 8

Custom EDI Schemas

Kofax Import Connector supports customized EDI schemas. The folder `C:\Program Files (x86)\Kofax\KIC-ED\KCPlugIn\EDI\Custom` created during KC Plug-In setup has the tools needed for customizing EDI schemas. Its subfolders (`config` and `spl`) contain the adjusted configuration files for Altova MapForce.

.NET Framework 4.0 has to be installed on the computer used for schema customizing.

Preparing Altova MapForce

1. Install Altova MapForce 2014 R2 Enterprise Edition.

   You can install it on the same computer as Kofax Import Connector, or on a different computer. In this example, we assume that everything is installed on a single computer.
2. Copy all files from the folder `C:\Program Files (x86)\Kofax\KIC-ED\KCPlugIn\EDI\Custom\config` to the `MapForceEDI` folder of your MapForce. Depending on your operating system and selected MapForce version, the destination folder is one of the following:
   - `Program Files\Altova\MapForce2014\MapForceEDI`
   - `Program Files (x86)\Altova\MapForce2014\MapForceEDI`
3. Copy all files from the folder `C:\Program Files (x86)\Kofax\KIC-ED\KCPlugIn\EDI\Custom\spl` to the MapForce `spl` folder. Depending on your operating system and selected MapForce version, the destination folder is one of the following:
   - `Program Files\Altova\MapForce2014\MapForceEDI\spl`
   - `Program Files (x86)\Altova\MapForce2014\MapForceEDI\spl`
4. In Altova MapForce, click **Tools** > **Options** > **Generation** tab and set **C# Settings** to **Microsoft Visual Studio 2010**.
5. In Altova MapForce, click **Connection** on the menu and select **Auto Connect Matching Children**.

Sample 1: HIPAA 837I, Customized for XYZ Acute Care

In the USA, the Health Insurance Portability and Accountability Act (HIPAA) defines the structure of electronic health care transactions. Health care providers typically issue a document called Companion Guide that states the HIPAA message versions they support and lists additional requirements or restrictions.

In this section, we assume that a fictitious company XYZ Acute Care uses Kofax Import Connector to send their received health care claim messages to Kofax Capture. According to their Companion Guide

document the messages follow the HIPAA standard 837I 5010 A2 (Health Care Claim Institutional). The EDI message in this example was extracted from the Companion Guide.

Verifying That EDI File Meets Standard

You can use Altova MapForce to verify that an EDI file meets the standard.

1. On the menu, click **Insert** > **EDI**.
2. Select an EDI collection (for example HIPAA.X12.5010A2) and the message type (for example 837-Q3 Health Care Claim: Institutional), then click **OK**.

   Note Altova MapForce installs only a subset of EDI collections. Click **Download additional EDI collections...** to access additional collections from Altova's website.

3. When prompted for a sample EDI file, click **Browse...** and select the EDI file you want to check.
4. Click **OK** in the Component Settings window.
   If the EDI file does not meet the standard, MapForce displays an error message:

   ```
   The newly assigned input file does not match the currently selected configuration file. Do
    you want to apply the sample file anyway?
   ```

5. Click **Yes**.
   MapForce lists all the errors found in the EDI file on the Messages pane, in the lower right section of the design window.

   Although the Companion Guide does not mention it, the sample message does not meet the standard, because a mandatory field is missing.

EDI Definitions in Altova MapForce

MapForce stores EDI collection definitions as text files with XML content. The files are located in the `MapForceEDI` folder of your MapForce installation. Each EDI collection has its own subfolder.

To analyze the problem from the previous section, look up the 837I definition (file 837I.config) from the subfolder `HIPAA.X12.5010A2`. This file describes the structure of the 837I message, which is rather complicated and consists of several nested loops.

The error message states that the field F1315 in segment CL1 of loop 2300 is missing. When you open 837I.config in an XML editor, you see that segment CL1 is defined as follows:

```
<Group name="Loop2300" maxOccurs="100" info="Claim Information" id="2300-837-Q3">
  <Segment name="CLM" info="Claim Information">
  <Segment name="DTP" minOccurs="0" info="Discharge Hour" nodeName="DTP_1">
  <Segment name="DTP" info="Statement Dates" nodeName="DTP_2">
  <Segment name="DTP" minOccurs="0" info="Admission Date/Hour" nodeName="DTP_3">
  <Segment name="DTP" minOccurs="0" info="Date - Repricer Received Date" nodeName="DTP_4">
  <Segment name="CL1" info="Institutional Claim Code">
    <Data ref="F1315" info="Admission Type Code" />
    <Data ref="F1314" minOccurs="0" info="Admission Source Code" />
    <Data ref="F1352" info="Patient Status Code" />
    <Data ref="F1345" minOccurs="0" maxOccurs="0" />
  </Segment>
```

The first field (F1315, Admission Type Code) is defined as mandatory.

Creating Custom EDI Definition in Altova MapForce

To make Altova MapForce accept your file, create a new EDI collection with a modified file 837I.config.

1. Copy the folder `HIPAA.X12.5010A2` to `HIPAA.XYZ`.
2. Remove the read-only attribute for the folder `HIPAA.XYZ`.
3. Delete unnecessary files from the folder `HIPAA.XYZ`. Keep only the following files:
   - ParserErrors.Config
   - X12.Codelist
   - EDI.Collection
   - X12.Segment
   - 837I.Config
   - Envelope.Config
4. Edit the file EDI.Collection in a text editor and delete all Message nodes except for the node with `File="837I.Config"`.

```
<?xml version="1.0" encoding="utf-8"?>
<Messages Version="3">
  <Meta>
    <Release>5010</Release>
    <Agency>X12</Agency>
  </Meta>
  <Root File="Envelope.Config" />
  <Message Type="837-Q3" File="837I.Config" Description="Health Care Claim:
 Institutional" />
</Messages>
```

5. Edit the file 837I.Config in a text editor and make the F1315 field optional (add `minOccurs="0"`).

```
<Data ref="F1315" minOccurs="0" info="Admission Type Code" />
```

6. Copy the folder `HIPAA.XYZ` to the `MapForceEDI` directory. You need administrator rights for action, because the destination folder is write-protected.

Now, when you browse EDI collections in Altova MapForce, you can find the newly defined collection HIPAA.XYZ with a single message type "837-Q3 Health Care Claim: Institutional". This modified EDI collection can be used to build the EDI type (schema and mapping files) for KC Plug-In.

Creating Schema and Sample File

1. Create a folder for custom EDI types, for example `D:\EDI\CustomSchemas`.
2. Open an administrator command prompt in the folder.
3. Run the CustomEdiToXsd batch job with the following syntax

```
CustomEdiToXsd <edipath> <targetpath> [<custom>]
```

`<edipath>` - directory containing the Altova MapForce EDI collection

`<targetpath>` - directory for Kofax Import Connector custom EDI schema

`<custom>` - optional sender identification; needed if Kofax Import Connector must process several variants of the same EDI message type

In this example, you don't need the optional parameter:

```
CustomEdiToXsd "C:\Program Files (x86)\Altova\MapForce2014\MapForceEdi\HIPAA.XYZ" D:\EDI
\CustomSchemas
```

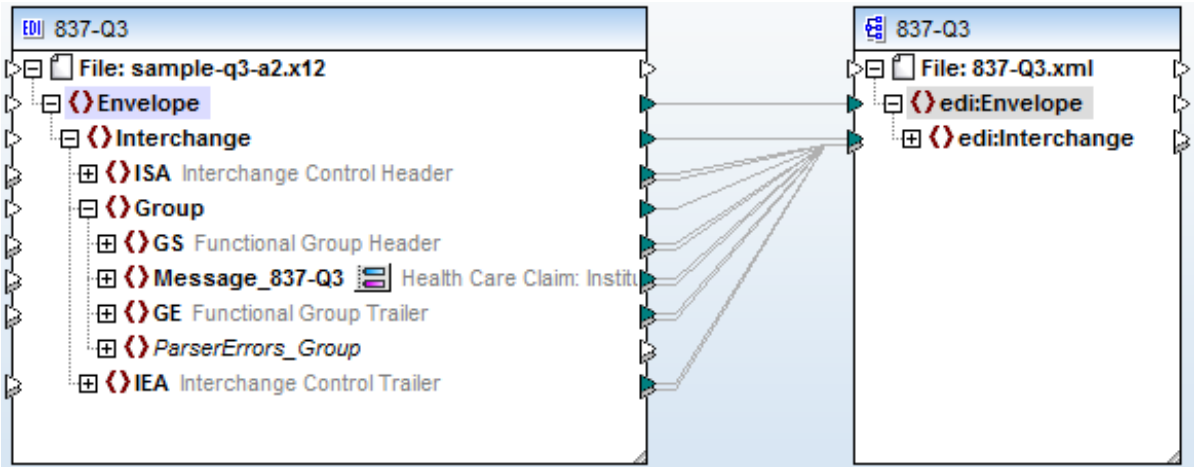
The batch job creates a new folder `HIPAA.XYZ` below the target folder, with the following files:

- 837-Q3.xsd
- info.xml
- Sample.xml
- X12.xsd

Sample.xml and the xsd files are part of the resulting XML type. Info.xml is needed for mapping and it will be deleted afterwards.

Creating EDI to XML Mapping

1. Start Altova MapForce and insert the EDI message type 837-Q3 from HIPAA.XYZ (no sample file needed).
2. On the menu, click **Insert** > **XML Schema/File** and select the newly created 837-Q3.xsd file.
3. When prompted to provide a sample file, click **Skip**.
4. Select Envelope as the root element.
5. In the Altova MapForce design window, use mouse to draw a connection line from Envelope on the left side to Envelope on the right side. MapForce connects matching child nodes automatically.



6. On the menu, click **File** > **Generate code in** > **C# (Sharp)**.
7. As a target directory, select `D:\EDI\CustomSchemas\HIPAA.XYZ\837-Q3`.
8. Click **OK** to create the source code.
   The Messages area shows when the source code is created.
9. Close MapForce. No need to save anything.
10. Open an administrator command prompt in the folder `D:\EDI\CustomSchemas\HIPAA.XYZ\837-Q3`.
11. Run the batch job compile.bat with the folder as a parameter:

    ```
    compile.bat D:\EDI\CustomSchemas\HIPAA.XYZ\837-Q3
    ```

    Visual Studio is not necessary for the task, the batch job uses tools included in .NET Framework. Source code files are deleted and replaced with an executable, mapping.exe.

You now have all files needed for importing the 837-Q3 folder into KC Plug-In as a new EDI type.

Sample 2: Customized Edifact 2010A ORDERS Message

Altova MapForce tutorial includes a section about Edifact schema customization. In this example, a new field is added to the CTA segment.

The tutorial explains that this can be done either globally (in the EDSD.segment file) or inline (in the ORDERS.config file). The result of the inline customization approach is available in the EDIFACT.Nanonull.zip file installed in the `MapForceExamples\Tutorial` folder (below `ProgramData`).

Creating Custom EDI Definition in Altova MapForce

1. Adjust the configuration files of Altova MapForce as instructed in the tutorial.
2. Copy the result to a subfolder under `MapForceEDI`, for example `MapForceEDI\EDIFACT.Nanonull`.

Now, when you browse EDI collections in Altova MapForce, you can find the newly defined collection EDIFACT.Nanonull with a single message type "ORDERS Purchase order message". This modified EDI collection can be used to build the EDI type (schema and mapping files) for KC Plug-In.

Creating Schema and Sample File

1. Create a folder for custom EDI types, for example `D:\EDI\CustomSchemas`.
2. Open an administrator command prompt in the folder.
3. Run the CustomEdiToXsd batch job with the following syntax

```
CustomEdiToXsd <edipath> <targetpath> [<custom>]
```

`<edipath>` - directory containing the Altova MapForce EDI collection

`<targetpath>` - directory for Kofax Import Connector custom EDI schema

`<custom>` - optional sender identification; needed if Kofax Import Connector must process several variants of the same EDI message type

In this example, let's use the optional sender identification, assuming that we receive such messages only from a business partner with sender identification " 003897733":

```
CustomEdiToXsd "C:\Program Files (x86)\Altova\MapForce2014\MapForceEdi\EDIFACT.Nanonull"
 D:\EDI\CustomSchemas 003897733
```

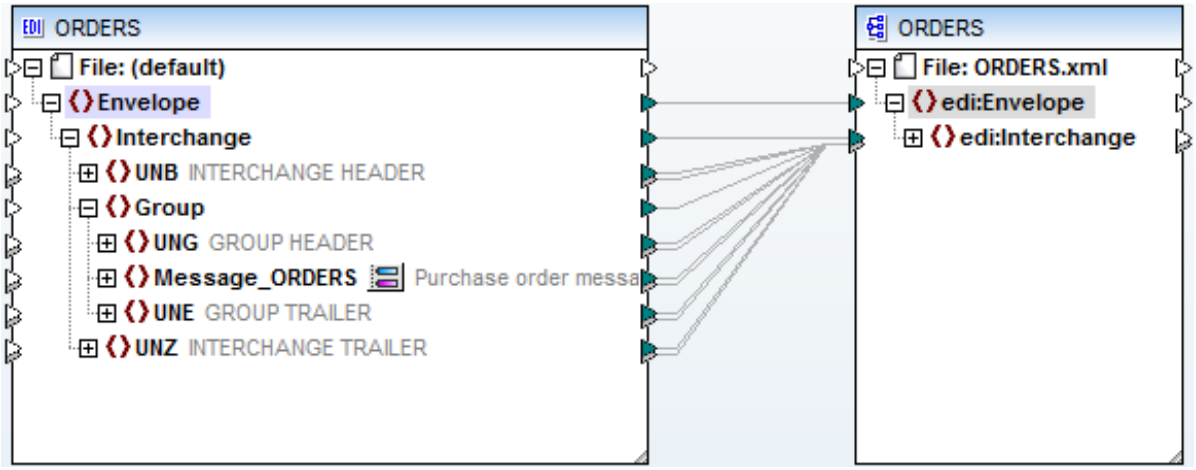
The batch job creates a new folder `EDIFACT.Nanonull` below the target folder, with the following files:

- Admin.xsd
- EDSD.xsd
- info.xml
- ORDERS.xsd
- Sample.xml

Sample.xml and the xsd files are part of the resulting XML type. Info.xml is needed for mapping and it will be deleted afterwards.

Creating EDI to XML Mapping

1. Start Altova MapForce and insert the EDI message type ORDERS from EDIFACT.Nanonull (no sample file needed).
2. On the menu, click **Insert** > **XML Schema/File** and select the newly created ORDERS.xsd file.
3. When prompted to provide a sample file, click **Skip**.
4. Select Envelope as the root element.
5. In the Altova MapForce design window, use mouse to draw a connection line from Envelope on the left side to Envelope on the right side. MapForce connects matching child nodes automatically.



6. On the menu, click **File** > **Generate code in** > **C# (Sharp)**.
7. As a target directory, select `D:\EDI\CustomSchemas\EDIFACT.Nanonull\ORDERS`.
8. Click **OK** to create the source code.
   The Messages area shows when the source code is created.
9. Close MapForce. No need to save anything.
10. Open an administrator command prompt in the folder `D:\EDI\CustomSchemas\EDIFACT.Nanonull\ORDERS`.
11. Run the batch job compile.bat with the folder as a parameter:

    ```
    compile.bat D:\EDI\CustomSchemas\EDIFACT.Nanonull\ORDERS
    ```

    Visual Studio is not necessary for the task, the batch job uses tools included in .NET Framework. Source code files are deleted and replaced with an executable, mapping.exe.

You now have all files needed for importing the ORDERS folder into KC Plug-In as a new EDI type.

Glossary