Kofax Communication Server

Capture Connector Developer's Guide

Version: 10.3.0

Date: 2019-12-13

KOFAX

Legal Notice

Table of Contents

Chapter 1: Preface **5**

Chapter 2: XML Mapping **6**

Using Sample Files 8

Importing Structured Data via Simple XML Mapping 9

Importing Structured Data via XML Import Connector-Compatible Mapping 12

Importing Structured Data via Generic Mapping 13

Generic Mapping 14

Creating XSL Transformations Manually 17

Create XSL Transformations Manually 18

Chapter 3: Web Services Interface for Kofax Monitor **20**

GetAllStates Function 20

GetConnection Function 21

GetConnectionNames Function 21

GetFeatureLicenseState Function 22

TestConversionErrors Function 22

Chapter 4: Scripting Interface **25**

Adding Additional References to Your Scripts 26

IbatchNameFormatter Interface Definition 26

IDocumentScript2 Interface Definition 28

Rejecting Messages from Script 31

Ignoring Messages from Script 31

Using BeforeDocumentImport to Access the Current Attachment 31

Using BeforeDocumentImport to get the EML/MSG/ZIP filename 34

IReRouteScript Interface Definition 34

IDocumentConverterScript Interface Definition 36

Append message body to attachments 41

Extract zip files 42

Convert documents to PDF or TIFF 43

Concatenate PDF files 44

Combine password protected PDF files 45

Convert password protected PDF files 47

Chapter 5: Custom Conversion Script **49**

Configuring Custom Conversion Script 49

Sample Script 49

Chapter 6: Custom Storage Strings.....50

Chapter 1

Preface

This document offers information about the following advanced topics related to Kofax Communication Server – Capture Connector:

- Examples for using XML mapping
- Web service interface used in Capture Connector
- Scripting
- Message metadata

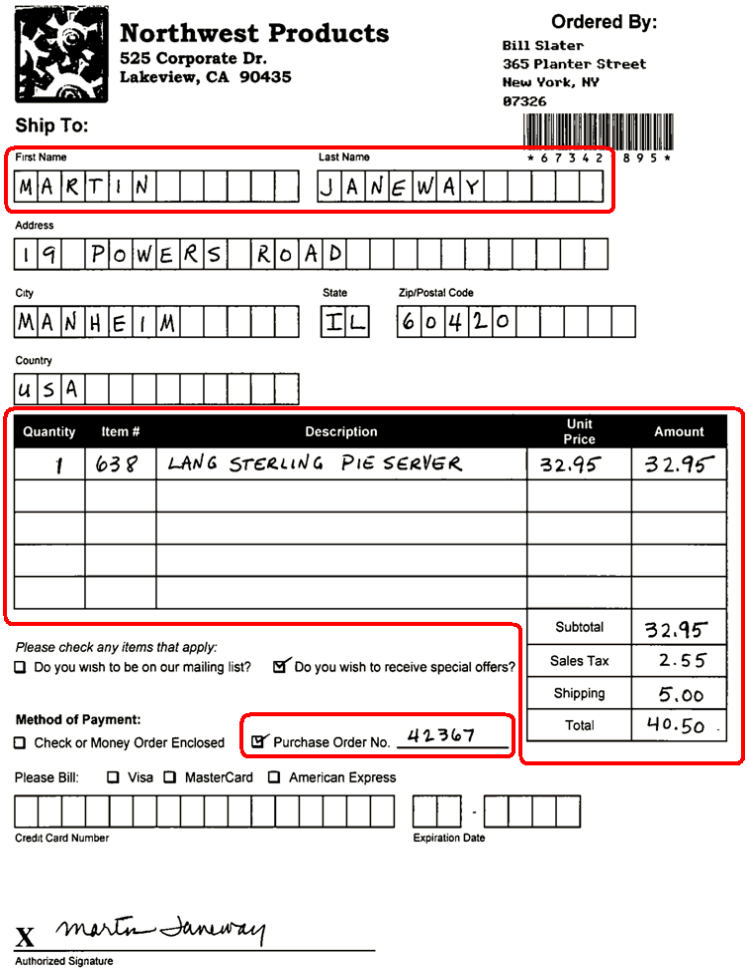
Chapter 2

XML Mapping

This chapter uses the sample files installed along with the Kofax Capture Plug-In to describe 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 called "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.

The examples in this chapter describe how to achieve this goal using simple and generic XML mapping. For the sake of simplicity, we select 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

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:

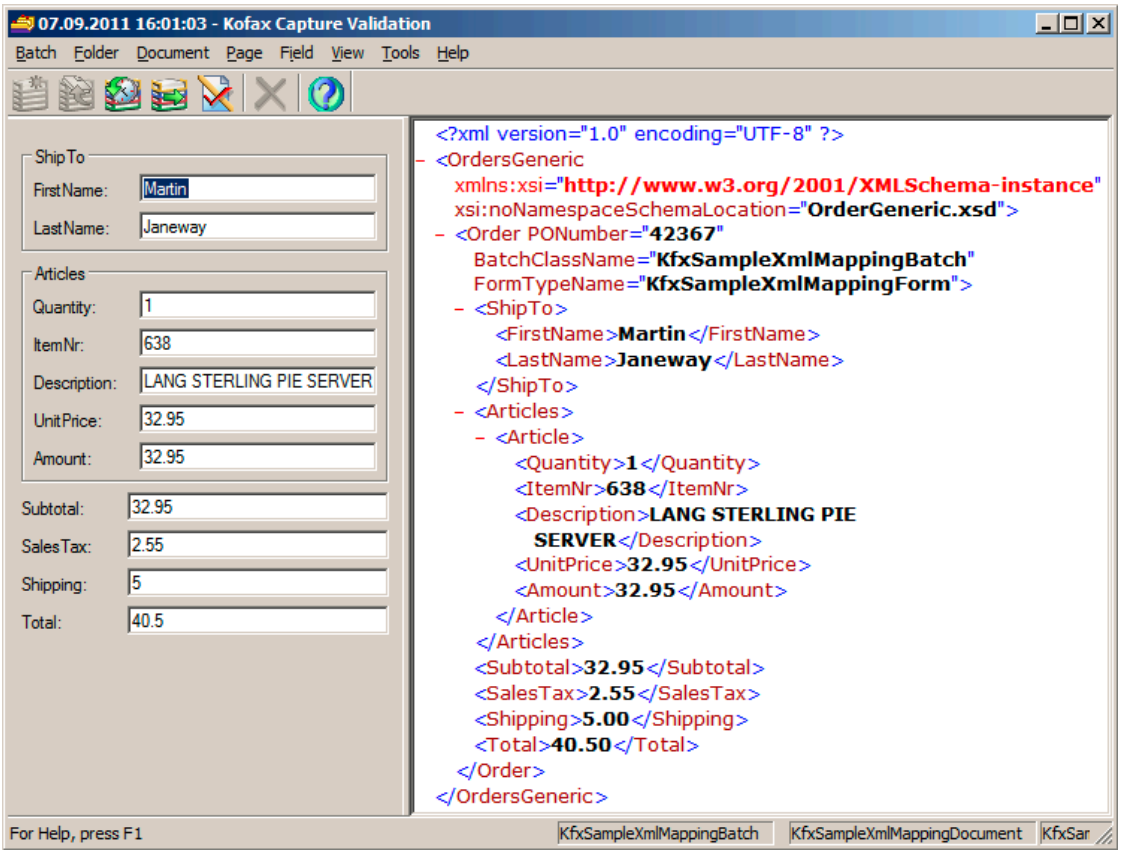
- Received XML data structure is always imported.
- Optionally, TIF images sent along with the structured XML data should be also imported.

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 the configuration section of the *Kofax Communication Server Capture 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 Kofax Capture 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 the Kofax Capture Plug-In where the mapping should occur. Configure it accordingly (don't forget to set the 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 built-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 may look similar to 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>
```

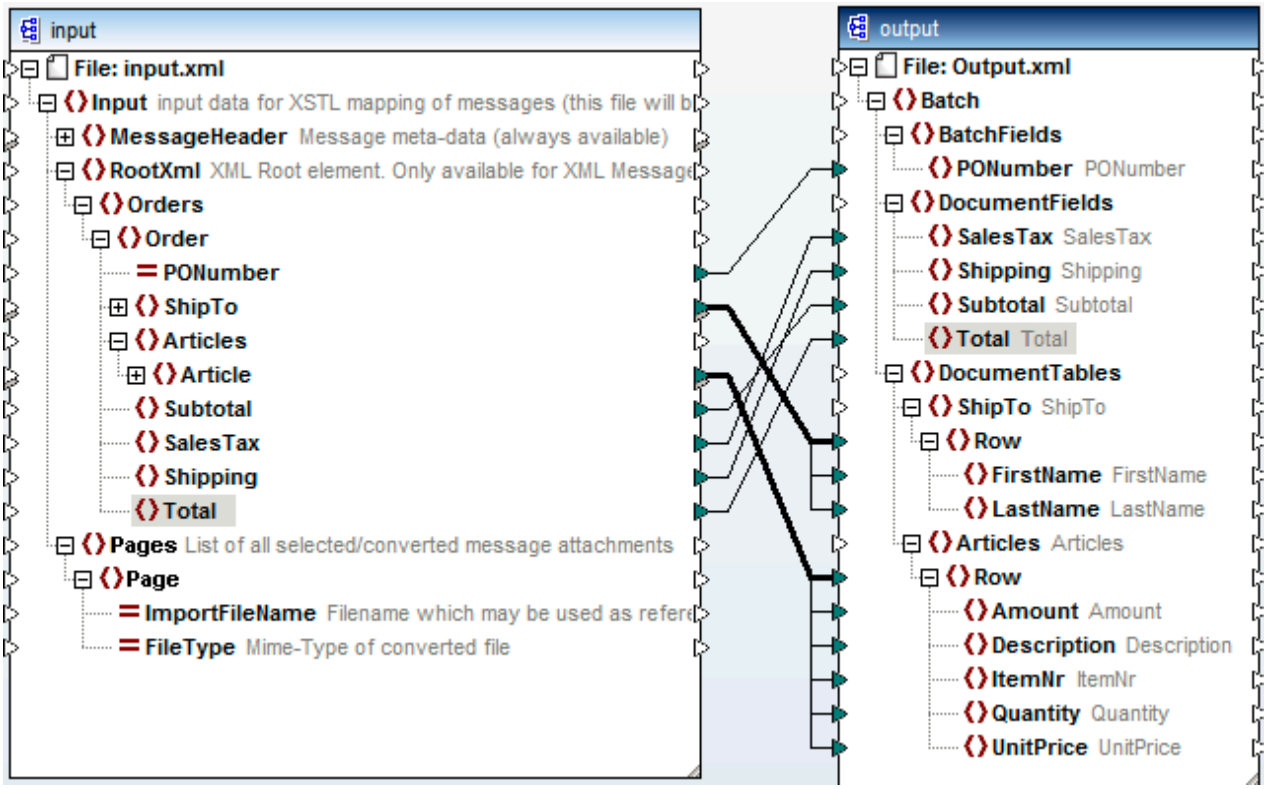
The schema file may look similar to this:

```
<?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, and 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

Northwest Products may decide to model their input data using the standard Kofax Capture XML Import Connector format. To do so, the XML input data would look similar to 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 the 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

Northwest Products may decide 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. For example, 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 the Kofax Capture Plug-In afterward. Alternatively, if you do not have MapForce, use the sample files from `Samples\DemoMapping\GenericMapping\SampleGenericMapping`. Click "Show files for Visual Designer" and copy all sample files to this folder.

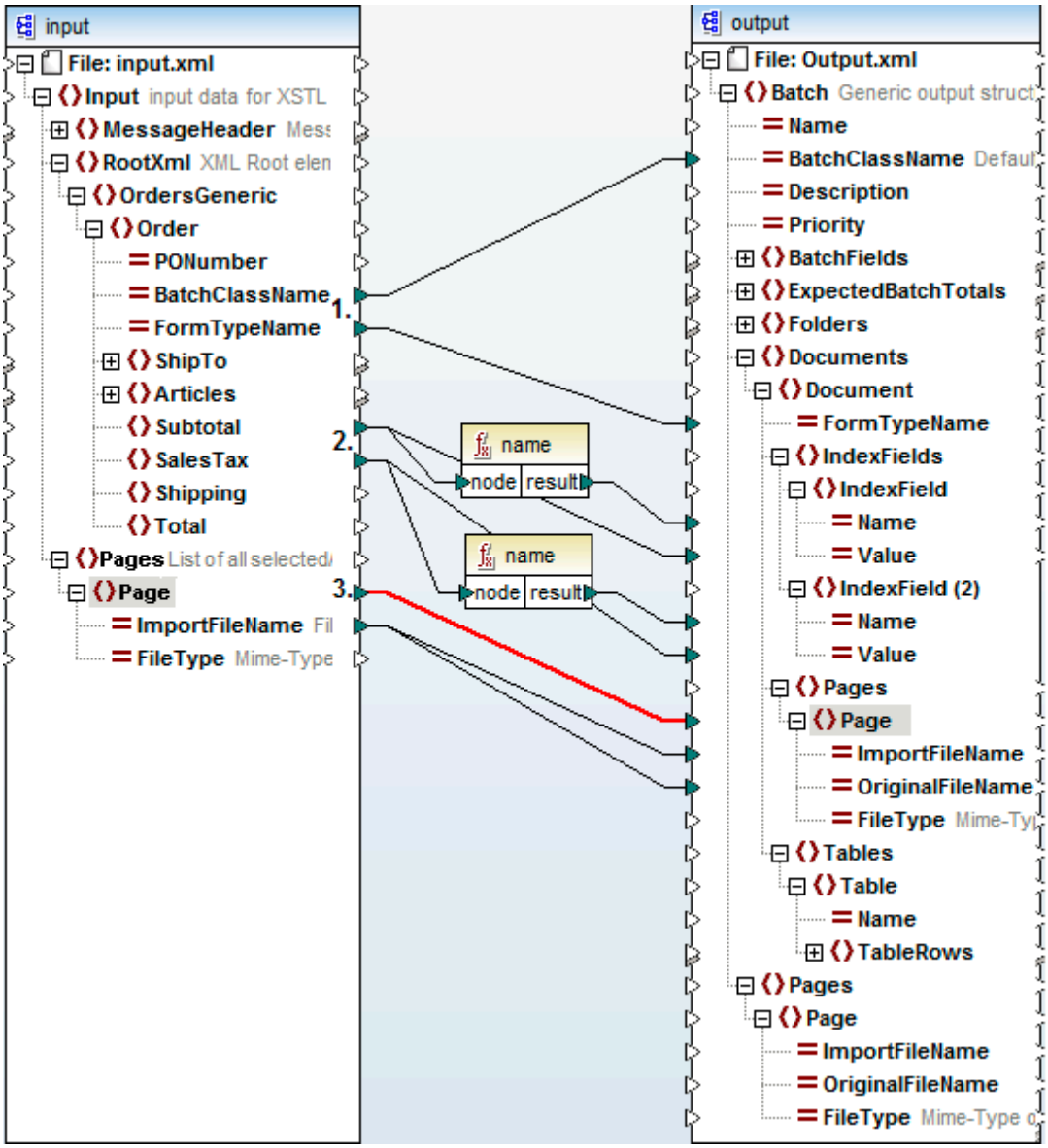
Generic Mapping

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 sample 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/attribute name would control the name, and its value the value attribute in the corresponding position in the output.xml (see 2 in the screen 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 (such as 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. For example, see how source elements “Subtotal” and “SalesTax” are mapped in 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 sample.

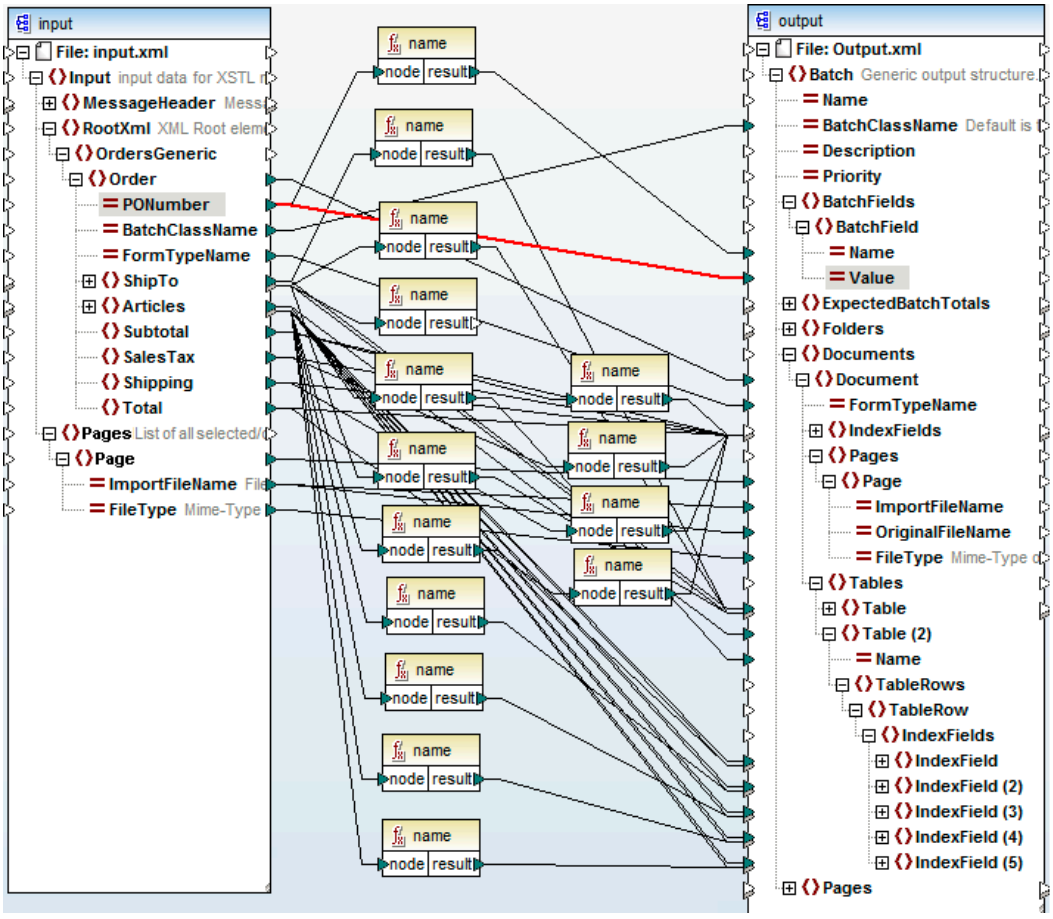


During the generation of the mapping in MapForce, it is possible to see how the sample XML document (of the destination's XML type) would be mapped using the current mapping. In our case it would look like this:

```
<?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. Northwest Products asks their XSLT expert to create the XSLT file 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 sample 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 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.

Create XSL Transformations Manually

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 an 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 may 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>
```

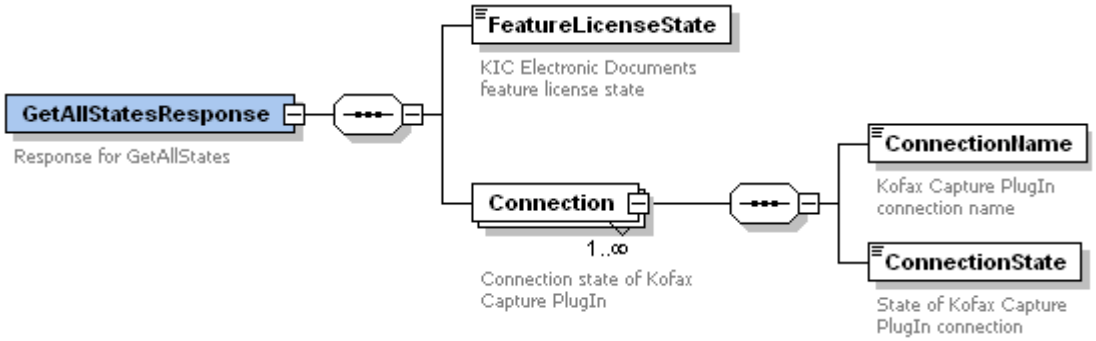
Chapter 3

Web Services Interface for Kofax Monitor

The Kofax Communication Server – Capture Connector – Import module offers web service interface functions for use with Kofax Monitor.

GetAllStates Function

This function returns the status of Capture 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.

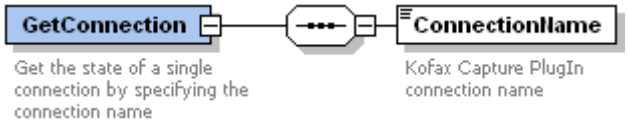
Example:

```
<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/KICElectronicDocuments">
    <FeatureLicenseState>1</FeatureLicenseState>
```

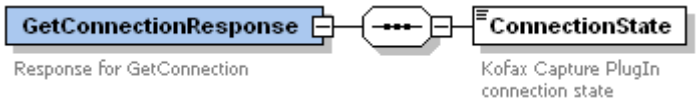
```
    <Connection>
      <ConnectionName>Connection2</ConnectionName>
      <ConnectionState>1</ConnectionState>
    </Connection>
    <Connection>
      <ConnectionName>Connection3</ConnectionName>
      <ConnectionState>0</ConnectionState>
    </Connection>
  </GetAllStatesResponse>
</s:Body>
```

GetConnection Function

This function returns the status of a connection between Capture Connector and TWS.



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.

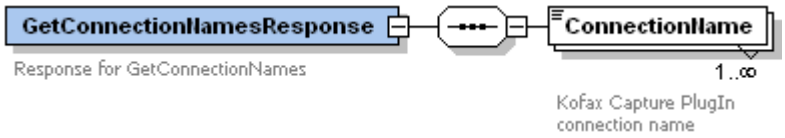
Example:

```
<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/KICElectronicDocuments">
    <ConnectionState>-1</ConnectionState>
  </GetConnectionResponse>
</s:Body>
```

GetConnectionNames Function

This function returns the names of all connections between Capture Connector and TWS.

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

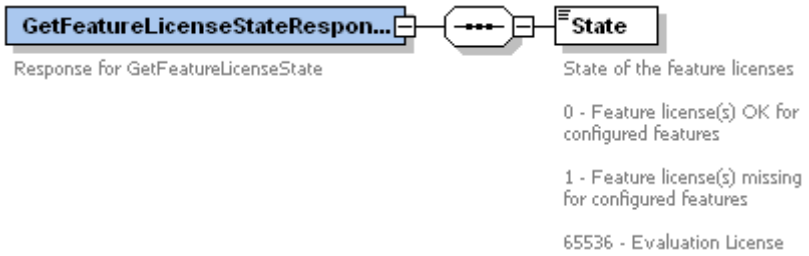


Example:

```
<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/KICElectronicDocuments">
    <ConnectionName>Connection2</ConnectionName>
    <ConnectionName>Connection3</ConnectionName>
  </GetConnectionNamesResponse>
</s:Body>
```

GetFeatureLicenseState Function

This function returns the status of Capture Connector feature licenses.

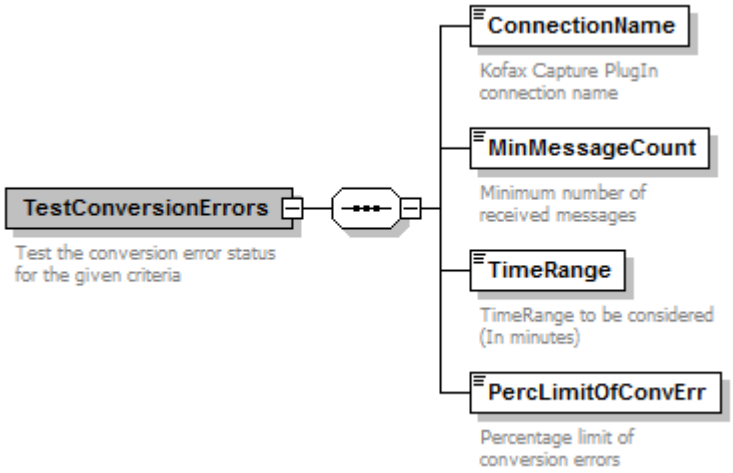


Example:

```
<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/
KICElectronicDocuments">
    <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.

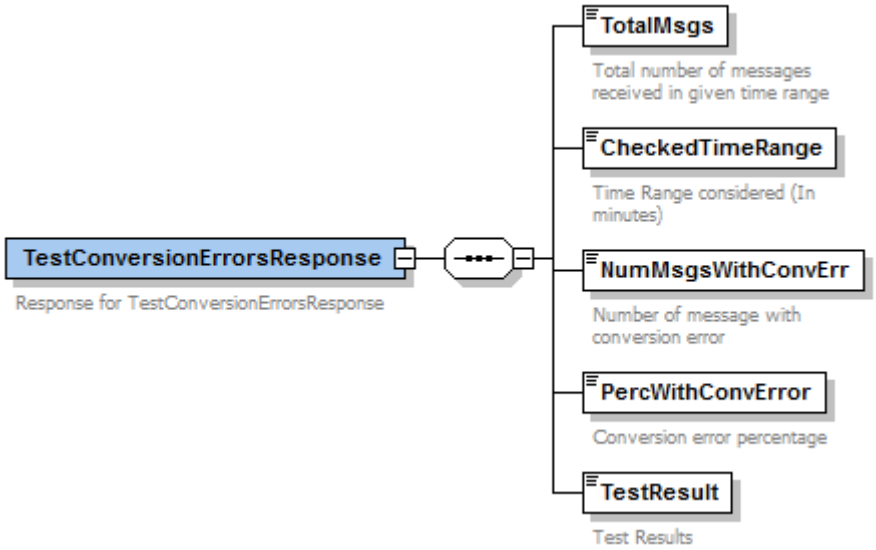


The input request fields are defined in the table below.

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 Capture Connector 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>
```

Chapter 4

Scripting Interface

Kofax Communication Server – Capture Connector 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 for example 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 TWS that a message has been rejected.

The Kofax Communication Server – Capture Connector 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.

With the connector, we provide a scripting DLL and a sample Visual Studio 2008 project, which enables you to create, run and test your scripts in Visual Studio before you deploy them to the Kofax Communication Server – Capture Connector. However, you can also create your scripts in any text editor.

The sample project is located in:

- `C:\Program Files\Kofax\KCS\Capture Connector\Scripting (assuming default installation on 32-bit operation systems)`
- `C:\Program Files (x86)\Kofax\KCS\Capture Connector\Scripting (assuming default installation on 64-bit operation systems)`

The scripts, though written in C#, need not to be compiled. You just have to configure the path to your script source file in the Kofax Communication Server – Capture Connector configuration. When the connector 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 the script contains errors, the connector will log the error and the default batch naming configured in the batch class will be 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 the message is imported into Kofax Capture. In the script you can make any changes to the document content.

Adding Additional References to Your Scripts

To add a reference to an assembly that is already in the 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 a line beginning with //ref: followed by the full paths to your DLLs delimited by a comma as a second line to your script. For example:

```
//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 the batch. With a single message batch, 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 in using this interface. You can use any of the defined Kofax Capture variables, such as “{Sequence Number}”. Kofax Capture variables are translated to Kofax Capture values (See the Kofax Capture documentation for more information about Kofax Capture variables).

The returning string from the function will be the actual batch name.

Important 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
DestConfig	Object (Destination Config)	The configuration of the destination to which the message has been redirected. Do NOT modify the properties of this object!

<table>
<tr><th>Property Name</th><th>Type</th><th>Description</th></tr>
<tr><td>Fields</td><td>IDictionary<string, string></td><td>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 Kofax Communication Server message fields listed below and any KCS Extension fields if they are defined in TWS.
See TWS documentation for more information about KCS Extension Fields.
List of standard available fields:
KcsOriginatorService
KcsOriginatorNumber
KcsOriginatorAddressBook
KcsOriginatorShortName
KcsOriginatorName
KcsOriginatorTitle
KcsOriginatorCompany
KcsOriginatorDepartment
KcsOriginatorFreeText
KcsOriginatorCorrel1
KcsOriginatorCorrel2
KcsOriginatorCorrel3
KcsOriginatorCorrel4
KcsRecipientService
KcsRecipientNumber
KcsRecipientAddressBook
KcsRecipientShortName
KcsRecipientsTo
KcsRecipientsCc
KcsRecipientsBcc
KcsRecipientServicesTo
KcsRecipientServicesCc
KcsRecipientServicesBcc
KcsRecipientName
KcsRecipientTitle
KcsRecipientCompany
KcsRecipientDepartment
KcsRecipientFreeText
KcsRecipientCorrel1
KcsRecipientCorrel2
KcsRecipientCorrel3
KcsRecipientCorrel4
KcsMessageCorrelation
KcsMessageID
KcsMessageOwnerReference
KcsMessageCustomField1
KcsMessageCustomField2
KcsMessageCustomField3
KcsMessageCustomField4
KcsMessageDeliveryOptions
KcsMessageDeliveryTermination
KcsMessageDeliveryIntendedSendTime
KcsMessageSendingCopy
KcsMessageFaxResolution
KcsMessageArchive
KcsMessageRegisteredMsg
KcsMessageLatestDeliveryTime
KcsMessageNamePosted
KcsMessageFileName
KcsMessageSubject
MessageEntryReceptionTimeStart
MessageEntryReceptionTimeEnd
KcsMessageReceptionTimeCreated
KcsMessagePages
KcsMessageDeliveryType
KcsMessageDeliveryPriority
KcsMessageDeliveryCostCenter
KcsMessageDeliverySuspectedDuplication
KcsMessageDeliveryErrorCode
KcsMessageDeliveryNote
KcsMessageDeliveryCost</td></tr>
</table>

Property Name	Type	Description
QueueName	string	The name of the queue from which the message was retrieved
MessageId	string	The KCS message ID
MessageFileName	string	The KCS message file name

Description of selected message fields:

Name	Description
KcsRecipientsTo	List of all To recipient numbers delimited by semicolon
KcsRecipientsCc	List of all Cc recipient numbers delimited by semicolon
KcsRecipientsBcc	List of all Bcc recipient numbers delimited by semicolon
KcsRecipientServicesTo	List of all service names and To recipient numbers delimited by semicolon in format: SERVICE,NUMBER;SERVICE,NUMBER
KcsRecipientServicesCc	List of all service names and Cc recipient numbers delimited by semicolon in format: SERVICE,NUMBER;SERVICE,NUMBER
KcsRecipientServicesBcc	List of all service names and Bcc recipient numbers delimited by semicolon in format: SERVICE,NUMBER;SERVICE,NUMBER

IDocumentScript2 Interface Definition

This is the definition of the IDocumentScript2 interface.

```
public interface IDocumentScript2
    {
/// <summary>
        /// Called by KCS Capture Connector before the import content and import order
 is determined
        /// Body and attachment content and import order can be maniuplated in this
 function
        /// </summary>
        /// <param name="messageBody">A readonly instance of the message being
 imported</param>
        /// <param name="messageBody">The list of message bodies received from KCS.</
param>
        /// <param name="attachments">The list of attachments received from KCS.</
param>
        /// <param name="extension">Reserved for future use.</param>
        void ManageMessageFiles(ReadonlyMessage message,
                                List<Attachment> messageBody,
                                List<Attachment> attachments,
                                object extension);
        /// <summary>
        /// Called by KCS Capture Connector 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 message bodies received from KCS.</
param>
        /// <param name="attachments">The list of attachments received from KCS.</
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 by KCS Capture Connector 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 message bodies received from KCS.</
param>
        /// <param name="attachments">The list of attachments received from KCS.</
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 will 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 a 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 will run before each Kofax Capture document is created.

If defined, the BeforeMessageImport function implementation will run 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, it 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, it 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, it 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 will be 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
ActualType	String	Type of the attachment set by TWS. Possible values: TEXT, BINARY_IMAGE, BINARY_nonIMAGE, ROOT_XML.
CaptureFileName	String	Empty. This field is populated only after the message is imported into Kofax Capture. For internal use only.
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.
ContentDisposition	String	MIME content disposition. If not set by TWS, the value is null.
ContentID	String	MIME content ID. If not set by TWS, the value is null.
ContentType	String	MIME type of the original attachment. If not set by TWS, the Capture Connector will set it depending on the file extension of the attachment.
CreationDate	DateTime	MIME creation date of original attachment. If not set by TWS, the value is 01.01.0001 00:00:00.
DocConversionError	String	If there was a document conversion error for this document, this property will contain the description of the error.
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.
Extension	String	The extension of the received file.
HierarchicalPosition	String	KCS hierarchical position of the file.
IsOriginal	Bool	Indicates if a file is an original file or a converted file.
IsOriginalEml	Bool	Reserved for internal use (Do not modify).
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.
IsXmlRendering	Bool	Indicates that the attachment is the result of a converted XML document.

Property name	Type	Description
LongOrBinaryFileName	String	The file name of the attachment/body received from Kofax Communication Server. 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.
ModificationDate	DateTime	MIME modification date of the original attachment. If not set by TWS, the value is 01.01.0001 00:00:00.
OriginalFileName	String	Contains the original file name of the attachment with the original extension (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 that 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.

Parameter	Description
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];

"Create document per attachment" UI Option	CurrentImport	Message Type	CurrentAttachment
Not applicable	`ImportData .PAGE_LIST`	Page list	-1 Current page list can be accessed using following code: `List<Attachmet> currPageList = new List<Attachment>(); for (int i = 0; i < info.currentPageList. Count; i+ +) currPageList.Add(attach- ments [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\KCS\Capture Connector\Scripting\ScriptSample) to a local directory.
2. Open **Additional settings** tab in KCC **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 the **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 KCC 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.

IReRouteScript Interface Definition

This script is called after Capture Connector receives a document from TWS, 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 TWS. The individual documents can be subsequently reimported to Capture Connector.

```
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 selected in the script is used and cannot change. Rules are ignored. It assumes that the user called the ReRoute function from the script. No refetch is performed.
ContinueWithRefetch	The destination selected in the script is used and cannot change. Rules are ignored. It assumes that the user called the ReRoute function from the script. Any changes to the message from the script are discarded, and the complete message is again fetched from the storage and 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. To import EDI documents in EML or MSG format, you have to configure it in the routing destination. The default EDI script is using ReSelectDestinationWORefetch as the 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, and 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.

Note SplitDocs option is not supported in Capture Connector.

"IReRouteScript" is the main interface. However, 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 such as 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\KCS\Capture Connector\Scripting\.`

Kofax Import Connector includes the script EDIReRouteScriptFile.cs, located in `C:\Program Files (x86)\Kofax\KCS\Capture Connector\Scripting\`. 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.

IDocumentConverterScript Interface Definition

The IDocumentConverterScript script interface extends the functionality of IReRouteScript script. IDocumentConverterScript exposes DocumentConverter() API along with ReRoute() API.

Following is the definition of the IDocumentConverterScript interface:

```
public interface IDocumentConverterScript : IReRouteScript
    {
        /// <summary>
        /// DocumentConverter API called by KCS.
        /// </summary>
        /// <param name="message">The incoming Message</param>
        /// <param name="converter">IDocumentConverter interface object used to
 Convert/Combine/Unzip the desired documents</param>
        /// <returns>The result of Convert/Combine/Unzip operation.</returns>
        ConversionResult DocumentConverter(Message message, IDocumentConverter
 converter);
    }
```

Description of the parameters

Parameter	Description
Message	The incoming Message
Converter	IDocumentConverter interface object used to convert/combine/unzip the desired documents.

Following is the definition of the IDocumentConverter API:

```
   /*
    * IDocumentConverterScript interface extends the functionality of IReRouteScript
 script.
    * IDocumentConverterScript exposes DocumentConverter() API along with ReRoute()
 API.
    * Using the implementation of IDocumentConverter interface in DocumentConverter()
 API,
    * we can achieve the following functionality:
    * 1) Combine Body of a message with individual Attachments. (Check
 AppendBodyToAttachments_FR7043.cs in KCSImportScriptingSample.zip)
    * 2) Convert any Documents to PDF and/or Tiff. (Check
 SampleDocumentConverterScript.cs in KCSImportScriptingSample.zip)
    * 3) Concatenate individual PDFs to a single PDF/A document.(Check
 SampleDocumentCombineScript.cs in KCSImportScriptingSample.zip)
    * 4) Extract Zipped archives.(Check SampleUnzipScript.cs in
 KCSImportScriptingSample.zip)
    * */
   /// <summary>
   /// IDocumentConverter Interface definition.
   /// </summary>
   public interface IDocumentConverter
   {
       /// <summary>
```

```
        /// This API is used to convert any Document to PDF and/or Tiff. For sample
 usage refer SampleDocumentConverterScript.cs in KCSImportScriptingSample.zip.
        /// </summary>
        /// <param name="output">If the operation is successful, output contains the
 converted document.</param>
        /// <param name="input">The document to be converted to TIFF or PDF.</param>
        /// <param name="destinationName">The name of the destination to be used to
 read the conversion settings.</param>
        /// <returns>The result of the conversion operation.</returns>
        ConversionResult Convert(out List<Attachment> output, Attachment input, string
 destinationName);
        /// <summary>
        /// This API is used to convert a List of input Documents to PDF
 and/or Tiff. For sample usage refer SampleDocumentConverterScript.cs in
 KCSImportScriptingSample.zip.
        /// </summary>
        /// <param name="output">If the operation is successful, output contains the
 converted documents.</param>
        /// <param name="input">The documents to be converted to TIFF or PDF.</param>
        /// <param name="destinationName">The name of the destination to be used to
 read the conversion settings.</param>
        /// <returns>The result of the conversion operation.</returns>
        ConversionResult Convert(out List<Attachment> output, List<Attachment> input,
 string destinationName);
        /// <summary>
        /// This API is used to convert a List of input Document to PDF
 and/or Tiff. For sample usage refer SampleDocumentConverterScript.cs in
 KCSImportScriptingSample.zip.
        /// </summary>
        /// <param name="output">If the operation is successful, output contains the
 converted documents.</param>
        /// <param name="input">The documents to be converted to TIFF or PDF.</param>
        /// <param name="converstionOptions">The options to be used for conversion.</
param>
        /// <returns>The result of the conversion operation.</returns>
        ConversionResult Convert(out List<Attachment> output, Attachment input,
 DocumentConversionOptions converstionOptions);
        /// <summary>
        /// This API is used to convert a List of input Documents to PDF
 and/or Tiff. For sample usage refer SampleDocumentConverterScript.cs in
 KCSImportScriptingSample.zip.
        /// </summary>
        /// <param name="output">If the operation is successful, output contains the
 converted documents.</param>
        /// <param name="input">The documents to be converted to TIFF or PDF.</param>
        /// <param name="converstionOptions">The options to be used for conversion</
param>
        /// <returns>The result of the conversion operation.</returns>
        ConversionResult Convert(out List<Attachment> output, List<Attachment> input,
 DocumentConversionOptions converstionOptions);
        /// <summary>
        /// This API is used to Unzip/Extract compressed files.
        /// </summary>
        /// <param name="output">The list of files in the compressed file.</param>
        /// <param name="input">The compressed file.</param>
        /// <param name="options">The options to be used for extraction.</param>
        /// <returns>The result of the extraction operation.</returns>
        ConversionResult Extract(out List<Attachment> output, Attachment input,
 DocumentExtractOptions options);
        /// <summary>
        /// This API is used to concatenate multiple PDF files into a single PDF or
 PDF/A file.
        /// </summary>
        /// <param name="output">The resultant concatenated PDF file.</param>
```

```
        /// <param name="input">The list of individual PDF files to be concatenated.</
param>
        /// <param name="options">The options to be used for the concatenate
 operation.</param>
        /// <returns>The result of the concatenation operation.</returns>
        ConversionResult CombineBinaries(out Attachment output, List<Attachment> input,
 DocumentCombineOptions options);
        /// <summary>
        /// This API is used to concatenate multiple PDF files into a single PDF or
 PDF/A file.
        /// </summary>
        /// <param name="output">The resultant concatenated PDF file.</param>
        /// <param name="body">The body of the message to be concatenated.</param>
        /// <param name="input">The list of attachments to be concatenated.</param>
        /// <param name="options">The options to be used for the concatenate
 operation.</param>
        /// <param name="appendBodyFirst">True, for body first and attachments. False,
 for attachments first and body.</param>
        /// <returns></returns>
        ConversionResult CombineBodyWithAttachments(out List<Attachment> output,
 Attachment body, List<Attachment> input, DocumentCombineOptions options, bool
 appendBodyFirst);
    }
            return result;
        }
        public eMessageScriptCode ReRoute(Message message, object extension)
        {
            return eMessageScriptCode.Other;
        }
    }
```

DocumentConverter () API details

Method name	Signature	Description
Convert	ConversionResult Convert(outList<Attachment> output, Attachment input, string destinationName);	Converts a document to PDF and/or TIFF on the basis of configuration in the destination defined by the destinationName parameter.
Convert	ConversionResult Convert(out List<Attachment> output, List<Attachment> input, string destinationName);	Converts a list of input documents to PDF and/or TIFF on the basis of configuration in the destination defined by the destinationName parameter.
Convert	ConversionResult Convert(out List<Attachment> output, Attachment input, DocumentConversionOptions converstionOptions);	Converts a list of input documents to PDF and/or TIFF on the basis of settings defined in the converstionOptions parameter.
Convert	ConversionResult Convert(out List<Attachment> output, List<Attachment> input, DocumentConversionOptions converstionOptions);	Converts a list of input documents to PDF and/or TIFF on the basis of settings defined in the converstionOptions parameter.
Extract	ConversionResult Extract(out List<Attachment> output, Attachment input, DocumentExtractOptions options);	Unzips or extracts compressed files and extracts portfolio PDF file.

Method name	Signature	Description
CombineBinaries	ConversionResult CombineBinaries(out Attachment output, List<Attachment> input, DocumentCombineOptions options);	Concatenates multiple PDF documents into a single PDF or PDF/A document. The Options parameter defines the settings to perform the combine operation. Note: This is only applicable for PDF documents.
CombineBodyWithAttachments	ConversionResult CombineBodyWithAttachments(out List<Attachment> output, Attachment body, List<Attachment> input, DocumentCombineOptions options, bool appendBodyFirst);	Concatenates the body of a message to each attachment in the message. • output: The resultant concatenated PDF file. • body: The body of the message. • input: The list of attachments to concatenate. • options: Options to use for the concatenate operation. • appendBodyFirst: If set to True, appends body first and then attachments. If set to False, appends attachments first and then body.

Class Properties

.Net SDK class name	Property name	Property type	Description and possible values
ConversionResult	Code	Integer	Sets or gets the error code of the conversion operation. '0' is returned for successful operation. Any other value implies a failed operation.
ConversionResult	Message	String	Contains the error message of a failed conversion operation. This remain empty for a successful conversion.
ConversionDetails	imageQuality	Integer	Sets the image quality for conversion. The value '100' is set for no compression. This is only applicable for grayscale and color TIFF images. Note: Compression may impact image quality.
ConversionDetails	imageResolution	Integer	Gets or sets the resolution of the output image. This corresponds to image DPI setting and is deduced from an Enum. The possible values are 200 and 300. Default is 200.

.Net SDK class name	Property name	Property type	Description and possible values
ConversionDetails	smoothFlags	Integer	Gets or sets a value for smoothing flags. Use these flags to configure the JPEG quality of the PDF to TIFF conversion. The possible values are: PDPageDrawSmoothText = 0x00000001 PDPageDrawSmoothLineArt = 0x00000002 PDPageDrawSmoothImage = 0x00000004 This parameter contains the result of an OR operation performed on the selected options.
ConversionDetails	renderFlags	Integer	Gets or sets the scaling property to use for conversion. The possible values are: 0: Disabled 1: Letter 2: Legal 3: A3 4: A4 5: A5 6: Match best 7: Match best Euro 8: Match best US
ConversionDetails	scaleTo	Enum	Gets or sets the output color. The possible values are: 1: Black and White 2: Grayscale 3: Color Default is 3.
ConversionDetails	imageCoding	Integer	
ConversionDetails	isALCFlatteningEnabled	bool	Gets or sets a value to use Adobe Life Cycle Server for flattening XFA forms. By default, flattening of XFA forms is disabled.
ConversionDetails	isPdfNormalationEnabled	bool	Gets or sets a value to normalize the document to PDF/A. By default, normalization is disabled.
DocumentCombineOptions	combineTo	Enum	Gets or sets a value to combine the documents. This is only applicable for PDF documents.

.Net SDK class name	Property name	Property type	Description and possible values
DocumentCombineOptions	WaitTimeSec	Integer	Gets or sets the wait time to complete the combine operation. Default is 120 seconds.
DocumentCombineOptions	conversionDetails	ConversionDetails	The ConversionDetails object contains the option to use for document conversion.
DocumentConversionOptions	Tif	Integer	Gets or sets the value to convert documents to TIFF or not. The possible values are: 0: Off 1: On
DocumentConversionOptions	Pdf	Integer	Gets or sets a value to convert to PDF or not. The possible values are: 0: Off 1: On
DocumentConversionOptions	WaitTimeSec	Integer	Gets or sets a value indicating the wait time to complete the combine operation.
DocumentConversionOptions	WaitTimeSecSpecified	bool	Gets or sets a value to identify whether the WaitTimeSec property is set or not.
DocumentConversionOptions	ConversionDetails	ConversionDetails	The ConversionDetails object contains the options to convert the document.
DocumentExtractOptions	Base64Passwords	List<string>	Gets or sets a list of base64 encoded passwords to open the PDF files. This class and property are reserved for future use.

The following features are available using the implementation of IDocumentConverterScript interface in DocumentConverter() API:

- Append body of a message to individual attachments
- Extract zipped archive files
- Convert documents to PDF and/or TIFF format
- Concatenate multiple PDF documents to a single PDF/A document

Append message body to attachments

This feature appends the body of a message to each message's attachment after the conversion. That is, each converted attachment has the message body appended.

Note This feature is only applicable for PDF conversion.

Sample code for using this function:

```
public class AppendBodyToAttachments_FR7043 : IDocumentConverterScript
   {
       public ConversionResult DocumentConverter(Message message, IDocumentConverter
converter)
       {
           ConversionResult result = new ConversionResult();
           if (converter != null)
           {
               List<Attachment> combinedAttachments = null;
               DocumentCombineOptions options = new DocumentCombineOptions();
               Attachment body = message.BodyAttachments.Find(att => att.IsBody ==
true && att.Extension.ToUpper() == "PDF");
               // Call CombineBodyWithAttachments API to combine Body to all the
attachments.
               result = converter.CombineBodyWithAttachments(out combinedAttachments,
body, message.BodyAttachments, options, false);
               if (result.Code == 0)
               {
                   message.BodyAttachments = combinedAttachments;
               }
           }
           return result;
       }
       public eMessageScriptCode ReRoute(Message message, object extension)
       {
       return eMessageScriptCode.Other;
       }
   }
```

To enable this, do the following:

1. Copy the script file AppendBodyToAttachments_FR7043.cs from the KCSImportScriptingSample.zip file. Default path:

   `C:\Program Files (x86)\Kofax\KCS\Capture Connector\Scripting`
2. Paste this script file to a local path. In Additional settings tab of KCC Destination Configuration, configure **Rerouting\Document conversion script path** field to browse the AppendBodyToAttachments_FR7043.cs file.
3. Ensure that the **Convert to PDF** option is selected in the **Import settings** tab.
4. Restart the KCC service.

Extract zip files

This feature extracts the zip files, that is, any email attachments or files in zip format are extracted. The extracted files can be converted to PDF or TIFF format.

Sample code for using this function:

```
public class SampleUnzipScript : IDocumentConverterScript
    {
        /// <summary>
        /// This method is used to Extract Zipped archives.
        /// </summary>
        /// <param name="message">The message to be imported into KC</param>
        /// <param name="converter">The Document Converter Interface object</param>
        /// <returns></returns>
        public ConversionResult DocumentConverter(Message message, IDocumentConverter
converter)
        {
```

```
            ConversionResult result = new ConversionResult();
            if (converter != null)
            {
                List<Attachment> unzippedAttachments = null;
                DocumentExtractOptions options = new DocumentExtractOptions();
                foreach (Attachment att in message.BodyAttachments)
                {
                    if (att.Extension.ToLower() == "zip" || att.ContentType.ToLower()
== "application/octet-stream")
                    {
                        result = converter.Extract(out unzippedAttachments, att,
options);
                        if (result.Code == 0 && unzippedAttachments != null &&
unzippedAttachments.Count > 0)
                        {
                            message.BodyAttachments.AddRange(unzippedAttachments);
                        }
                    }
                }
                return result;
            }
            return result;
        }
        public eMessageScriptCode ReRoute(Message message, object extension)
        {
            return eMessageScriptCode.Other;
        }
    }
```

To enable this feature, do the following:

1. Copy the script file SampleUnzipScript.cs from the KCSImportScriptingSample.zip file.

   Default path: `C:\Program Files (x86)\Kofax\KCS\Capture Connector\Scripting`
2. Paste this script file to a local path. In **Additional settings** tab of KCC **Destination configuration**, configure **Rerouting\Document conversion script path** field to browse the SampleUnzipScript.cs file.
3. Restart the KCC service.

Convert documents to PDF or TIFF

This feature converts the files to PDF or TIFF formats. If the PDF files are converted to PDF format, you can concatenate these files.

Sample code for using this function:

```
{
        /// <summary>
        /// This method is used to Convert dicuments to Tiff/PDF
        /// </summary>
        /// <param name="message">The message to be imported into KC</param>
        /// <param name="converter">The Document Converter Interface object</param>
        /// <returns></returns>
        public ConversionResult DocumentConverter(Message message, IDocumentConverter
converter)
        {
            ConversionResult result = new ConversionResult();
            if (converter != null)
            {
                List<Attachment> convertedAttachments = null;
                DocumentConversionOptions options = new DocumentConversionOptions();
```

```
                options.ConversionDetails = new ConversionDetails();
                // To Convert documents to Tiff or PDF, configure the required options
below.
                // Configuring Both PDF and Tif conversion.
                options.Pdf = options.Tif = 1;
                result = converter.Convert(out convertedAttachments,
message.BodyAttachments, options);
                if (result.Code == 0 && convertedAttachments != null &&
convertedAttachments.Count > 0)
                {
                    message.BodyAttachments.AddRange(convertedAttachments);
                }
            }
            return result;
        }
        public eMessageScriptCode ReRoute(Message message, object extension)
        {
            return eMessageScriptCode.Other;
        }
    }
```

To enable this feature, do the following:

1. Copy the script file SampleDocumentConverterScript.cs from the KCSImportScriptingSample.zip file. Default path:

   `C:\Program Files (x86)\Kofax\KCS\Capture Connector\Scripting`
2. Paste this script file to a local path. In **Additional settings** tab of KCC Destination configuration, configure **Rerouting\Document conversion script path** field to browse the SampleDocumentConverterScript.cs file.
3. Ensure that the **Import original content** option is selected in the **Import settings** tab.
4. Restart the KCC service.

Concatenate PDF files

This feature concatenates individual PDF documents to a single PDF/A document. The output contains individual PDF documents along with one concatenated PDF/A document.

Note This feature is only applicable for PDF conversion.

Sample code for using this function:

```
public class SampleDocumentCombineScript : IDocumentConverterScript
    {
        public ConversionResult DocumentConverter(Message message, IDocumentConverter
converter)
        {
            ConversionResult result = new ConversionResult();
            if (converter != null)
            {
                Attachment combinedAttachment = null;
                DocumentCombineOptions options =
                    new DocumentCombineOptions
                    {
                        /* Setting CombineTo to PDF.*/
                        combineTo = CombineOptionsCombineTo.PDF,
                        WaitTimeSec = 120,
                        conversionDetails = new ConversionDetails()
                    };
                // Enabling PDF normalization and ALC flattening.
```

```
                options.conversionDetails.isALCFlatteningEnabled = true;
                options.conversionDetails.isPdfNormalationEnabled = true;
                result = converter.CombineBinaries(out combinedAttachment,
message.BodyAttachments, options);
                if (result.Code == 0 && combinedAttachment != null)
                {
                    // Uncomment below line To replace all the files and import only
the combined attachment
                    //message.BodyAttachments.Clear();
                    // To add the combined Attachment to the list of files imported
                    message.BodyAttachments.Add(combinedAttachment);
                }
            }
```

To enable this feature, do the following:

1. Copy the script file SampleDocumentCombineScript.cs from the KCSImportScriptingSample.zip file. Default path:

   `C:\Program Files (x86)\Kofax\KCS\Capture Connector\Scripting`
2. Paste this script file to a local path. In **Additional settings** tab of KCC **Destination configuration**, configure **Rerouting\Document conversion script path** field to browse the SampleDocumentCombineScript.cs file.
3. Restart the KCC service.

Combine password protected PDF files

This feature unlocks password protected PDF files and concatenate the output files.

Sample code for using this feature:

```
class CombineEncryptedPdfs : IDocumentConverterScript
    {
        public ConversionResult DocumentConverter(Message message, IDocumentConverter
converter)
        {
            ConversionResult result = new ConversionResult();

            if (converter != null)
            {
                Attachment combinedAttachment = null;
                DocumentCombineOptions options =
                    new DocumentCombineOptions
                    {
                        /* Setting CombineTo to PDF.*/
                        combineTo = CombineOptionsCombineTo.PDF,
                        WaitTimeSec = 120,
                        conversionDetails = new ConversionDetails()
                    };

                // Enabling PDF normalization and ALC flattening.
                options.conversionDetails.isALCFlatteningEnabled = true;
                options.conversionDetails.isPdfNormalationEnabled = true;
                options.conversionDetails.PDFACompliance = PDFAComplianceType.PDFA3a;

                /*
                 * Passwords can be read from any source.
                 * They can be read from an XML file or a database and
                 * be used for unlocking the incoming PDF documents.
                */
```

```
                // As sample, we are hardcoding the passwords here.
                List<string> passwords = new List<string>();
                passwords.Add("GoodPassW0rd");
                passwords.Add("badpassword");
                passwords.Add("Just@Pwd");
                passwords.Add("RE@ll$TouGHP@sSW0R&");
                passwords.Add("test@88");
                passwords.Add("test");
                passwords.Add("abcd");

                // The passwords set here would be encrypted before sending the request
to
                // Message connector for unlocking the PDF files.
                options.conversionDetails.SetPasswords(passwords);

                // Call the Convert API with the documents which needs conversion as
input.
                List<Attachment> pdfAttachments = message.BodyAttachments.FindAll(att
=> att.Extension.ToUpper().Contains("PDF"));

                if (pdfAttachments.Count > 0)
                {
                    result = converter.CombineBinaries(out combinedAttachment,
pdfAttachments, options);
                }

                if (result.Code == 0 && combinedAttachment != null)
                {
                    // Uncomment below line to replace all the files and import only
the combined attachment
                    //message.BodyAttachments.Clear();

                    // To add the combined Attachment to the list of files imported
                    message.BodyAttachments.Add(combinedAttachment);
                }
            }

            return result;
        }

        public eMessageScriptCode ReRoute(Message message, object extension)
        {
            return eMessageScriptCode.Other;
        }
    }
```

To enable this feature, do the following:

1. Copy the script file CombineEncryptedPdfs.cs from the KCSImportScriptingSample.zip file. Default path:

   `C:\Program Files (x86)\Kofax\KCS\Capture Connector\Scripting`

2. In KCC:

   a. Paste this script file to a local path. In **Additional settings** tab of KCC Destination configuration, configure **Rerouting\Document conversion script path** field to browse the CombineEncryptedPdfs.cs.

   b. Ensure that the **Import original content** option is selected in the **Import settings** tab.

   c. Restart the KCC service.

Convert password protected PDF files

This feature unlocks password protected PDF files. The passwords for unlocking the PDF files can be fetched from any source, such as an XML file or a database.

Sample code for using this feature:

```
public class ConvertEncryptedPdf : IDocumentConverterScript
    {
        /// <summary>
        /// This method is used to Convert dicuments to Tiff/PDF
        /// </summary>
        /// <param name="message">The message to be imported into KC</param>
        /// <param name="converter">The Document Converter Interface object</param>
        /// <returns></returns>
        public ConversionResult DocumentConverter(Message message, IDocumentConverter
 converter)
        {
            ConversionResult result = new ConversionResult();

            if (converter != null)
            {
                List<Attachment> convertedAttachments = null;
                DocumentConversionOptions options = new DocumentConversionOptions();

                /* ConversionDetails class exposes the following settings:
                 *  PDF Normalization
                 *  Flattening portfolio PDFs using Adobe Life Cycle
                 *  Image Quality
                 *  Image Resolution
                 *  Image Scaling and Image Coding
                 *  Smoothening and Rendering Flags
                */

                options.ConversionDetails = new ConversionDetails();

                // Configuring PDF Normalization and Flattening portfolio PDFs.
                options.ConversionDetails.isPdfNormalationEnabled = true;
                options.ConversionDetails.isALCFlatteningEnabled = true;

                // To Convert documents to Tiff or PDF, configure the required options
 below.

                // Configuring PDF conversion.
                // NOTE: If the input is a PDF document, and PDF normalization is not
 enabled, then output will be empty.
                options.Pdf = 1;

                // Configuring Tif conversion.
                options.Tif = 1;

                /*
                 * Passwords can be read from any source.
                 * They can be read from an XML file or a database and
                 * be used for unlocking the incoming PDF documents.
                */

                // As sample, we are hardcoding the passwords here.
                List<string> passwords = new List<string>();
                passwords.Add("GoodPassW0rd");
                passwords.Add("badpassword");
```

```
                passwords.Add("Just@Pwd");
                passwords.Add("RE@ll$TouGHP@sSW0R&");
                passwords.Add("test@88");
                passwords.Add("test");
                passwords.Add("abcd");

                // The passwords set here would be encrypted before sending the request
to
                // Message connector for unlocking the PDF files.
                options.ConversionDetails.SetPasswords(passwords);

                // Call the Convert API with the documents which needs conversion as
input.
                List<Attachment> pdfAttachments = message.BodyAttachments.FindAll(att
=> att.Extension.ToUpper().Contains("PDF"));

                if (pdfAttachments.Count > 0)
                {
                    result = converter.Convert(out convertedAttachments,
pdfAttachments, options);
                }

                if (result.Code == 0 && convertedAttachments != null &&
convertedAttachments.Count > 0)
                {
                    // Add the converted documents to the message
                    // so that they will be imported into Kofax Capture.
                    message.BodyAttachments.AddRange(convertedAttachments);
                }
            }

            return result;
        }

        public eMessageScriptCode ReRoute(Message message, object extension)
        {
            return eMessageScriptCode.Other;
        }
```

To enable this feature, do the following:

1. Copy the script file ConvertEncryptedPdf.cs from the KCSImportScriptingSample.zip file. Default path:

   `C:\Program Files (x86)\Kofax\KCS\Capture Connector\Scripting`

2. In KCC:

   a. Paste this script file to a local path. In **Additional settings** tab of KCC Destination configuration, configure **Rerouting\Document conversion script path** field to browse the ConvertEncryptedPdf.cs.

   b. Ensure that the **Import original content** option is selected in the **Import settings** tab.

   c. Restart the KCC service.

Chapter 5

Custom Conversion Script

The document conversion function in TWS 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

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

1. In the TWS 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.
2. Save the configuration and restart TWS.
3. Create a batch file as shown below that performs the conversion to PDF.

   Save it as "CustomToPdf.bat" to the Scripts subfolder of the TWS installation directory.
4. Test the conversion with the "Convert Document" test page of the TWS 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 6

Custom Storage Strings

When you configure a destination in Capture Connector, 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 message.

Most custom storage strings are listed in section *IbatchNameFormatter Interface Definition*, in the Fields row of the table. Description for a subset of the strings is also available in the same section. In addition, the following strings are available.

Custom storage string	Description
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-DDTHHMM-SS.
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)
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**. PageNr is the page number of the first page of the related converted document in Kofax Capture. **Examples:** • 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