

Tungsten Capture Developer's Guide

Version: 11.2.0

Date: 2024-11-12

TUNGSTEN
AUTOMATION

© 2024 Tungsten Automation. All rights reserved.

Tungsten and Tungsten Automation are trademarks of Tungsten Automation Corporation, registered in the U.S. and/or other countries. All other trademarks are the property of their respective owners. No part of this publication may be reproduced, stored, or transmitted in any form without the prior written permission of Tungsten Automation.

Table of Contents

Preface	8
Related Documentation.....	8
<i>Tungsten Capture Installation Guide</i>	8
<i>Tungsten Capture Administrator's Guide</i>	8
<i>Tungsten Capture Help</i>	8
<i>Tungsten Capture API References</i>	8
<i>Tungsten Capture Release Notes</i>	9
Training.....	9
Getting help with Tungsten Automation products.....	9
Chapter 1: Overview	11
Reasons to Customize Tungsten Capture.....	11
How to Customize Tungsten Capture.....	11
API Libraries.....	12
Backward Compatibility.....	14
Fluent User Interface (Fluent UI) Customizations.....	15
Additional Resources.....	16
How to Implement Your Custom Scripts and Modules.....	17
Chapter 2: Custom Script Creation Using VB.NET	18
Software Requirements.....	18
Validation Script Creation in VB.NET.....	19
Selecting the Scripting Language.....	19
Tungsten Capture .NET Scripting API.....	19
VB.NET Project File Location.....	20
Deployment of a VB.NET Project.....	21
Script Publishing Requirements.....	21
Sample VB.NET Validation Script.....	21
Sample Validation Script Code.....	22
Testing VB.NET Custom Scripts.....	23
Error Handling in VB.NET.....	24
FatalErrorException.....	24
RejectAndSkipDocumentException.....	24
ValidationErrorException.....	24
Recognition Script Creation in VB.NET.....	25
Tungsten Capture .NET Scripting API.....	25
Creating a Recognition Script Using VB.NET.....	26

Debugging Your Settings with a VB.NET Recognition Script.....	27
Sample VB.NET Recognition Script.....	27
Removing a Recognition Script.....	28
Field script.....	28
Sample VB.NET Field Script.....	29
Chapter 3: Custom Script Creation Using Visual C#.....	30
Software Requirements.....	30
Validation Script Creation in Visual C#.....	31
Selecting the Scripting Language.....	31
Tungsten Capture Visual C# Scripting API.....	32
Creating a Custom Visual C# Validation Script.....	33
Visual C# Project File Location.....	34
Deployment of a Visual C# Project.....	35
Script Publishing Requirements.....	35
Testing Visual C# Custom Scripts.....	35
Error Handling in Visual C#.....	36
FatalErrorException.....	36
RejectAndSkipDocumentException.....	36
ValidationErrorException.....	36
Recognition Script Creation in Visual C#.....	37
Tungsten Capture Visual C# Scripting API.....	37
Creating a Recognition Script Using Visual C#.....	38
Debugging Your Settings with a Visual C# Recognition Script.....	39
Sample Visual C# Recognition Script.....	40
Removing a Recognition Script.....	40
Field script.....	41
Sample Visual C# Field Script.....	41
Chapter 4: Registration File Creation.....	43
Format for the Registration File.....	43
[Modules] Section.....	43
[Module Name] Section.....	44
[Workflow Agents] Section.....	48
[Workflow Agent Name] Section.....	49
[Setup Programs] Section.....	49
[Setup] Section.....	50
[Menu] Section.....	51
[Menu Bar] Section.....	52
Sample Registration Files.....	52

Using the Administration Module to Manage Extensions.....	56
Custom Module Management.....	56
Workflow Agent Management.....	57
Tungsten Capture Extension Registration Utility.....	58
Command Line Parameters.....	59
Input.....	61
Output.....	61
Chapter 5: Workflow Agent Creation.....	62
Workflow Agent Design.....	62
Setup OCX Implementation.....	62
Writing the Runtime Module.....	63
Code Project Settings.....	63
Workflow Agent Sample.....	64
Registration File Creation.....	68
Registering the Setup OCX.....	70
Registering a Setup OCX Not Associated with a Custom Extension.....	70
Installing and Registering the Workflow Agent.....	70
Removing a Workflow Agent.....	71
Chapter 6: Setup OCX Creation.....	72
Setup OCX Design.....	72
Writing a Setup OCX.....	72
Code Project Settings.....	73
Sample Setup OCX for the Custom Workflow Agent.....	73
Setup OCX for the Workflow Agent.....	73
Registering the Setup OCX.....	76
Registering a Setup OCX Not Associated with a Custom Extension.....	76
Setup OCX Registry Entries.....	76
Tab Registry Keys.....	78
Loading the Setup OCX.....	79
Unloading the Setup OCX.....	80
Setup OCX Panels.....	80
Enabling Panels.....	80
Context Menus.....	81
Enabling Context Menu Items.....	81
Ribbon.....	81
Custom Tab Names.....	81
Enabling/Disabling Custom Commands.....	82
Panels.....	82

Batch Class Publishing.....	82
Setup OCX Development API.....	83
Chapter 7: Custom Panels and Applications.....	84
Programming in a High Availability Environment.....	84
User Interface Design and Behavior.....	84
Custom Panels.....	85
Custom Tabs.....	86
Custom Panel Installation.....	87
Invoking Tungsten Capture Commands from a Custom OCX Panel.....	90
Sample Custom Panel.....	92
Chapter 8: Custom Module Creation.....	97
Custom Modules.....	97
High Availability Environments.....	97
Error Handling Guidelines.....	97
Sample Applications.....	99
Typical Development Tasks.....	99
Design the Custom Module.....	99
Create the Setup OCX.....	100
Write the Runtime Application.....	100
Create the Custom Module Registration File.....	100
Register the Custom Module.....	101
Create an Installation Program.....	101
Document Routing.....	101
Document Routing Functions.....	101
About Document Routing Features.....	103
Using Tungsten Transformation.....	104
Using the Sample Custom Module.....	104
Sample Custom Module.....	104
Chapter 9: Creating an Export Connector.....	108
Tungsten Capture Export Type Library.....	108
Tungsten Capture and the Export Process.....	109
Requirements for the Export Connector Setup.....	109
Requirements for the Export Connector.....	110
ReleaseSetupData and ReleaseData Objects.....	110
COM Servers: In-proc or Out-of-proc?.....	112
Registering Your Export Connector.....	112
Scripting in a High Availability Environment.....	113
Chapter 10: Deploying Customizations.....	114

Installing the Customization Deployment Service.....	115
Command Line Parameters.....	115
Setting Up a Customization Deployment.....	116
Initiating a Customization Deployment.....	117
Viewing Customization Deployment Status.....	118
Deploying Customizations While Applications Are Running.....	119
About the Customization Deployment Process.....	120
Administrator Actions.....	120
Deployment Service Actions.....	120
Tungsten Capture Network.....	121
Appendix A: Custom Module Sample.....	122
Licensing.....	123
Tungsten Capture Optimized Custom Module .NET Type Implementation Library.....	123
Tungsten Capture Document Access .NET Type Implementation Library.....	124
Development Environment.....	125
Setup OCX.....	125
Runtime Executable.....	125
Installing the Sample Custom Module.....	125
Registering the Sample Custom Module.....	126
Adding the Sample to the Batch Processing Workflow.....	126
Using Batch Manager with a Custom Module.....	127
Creating a Batch to Open in the Sample Custom Module.....	127
Processing by Document.....	128
Setting the Batch Custom Storage String.....	128
Getting the Batch Custom Storage String.....	129
XML Transport Files.....	129
Copying the XML Files Back to Tungsten Capture.....	129
Corrupt XML.....	130
Create Page.....	130
Appendix B: Workflow Agent Sample.....	131
Installing the Sample Workflow Agent.....	131
Registering the Sample Workflow Agent.....	132
Using the Sample Workflow Agent.....	132
Appendix C: Converting Script Code from SBL to VB.NET.....	134
Variables.....	134
Procedures and Functions.....	135

Preface

The *Tungsten Capture Developer's Guide* provides information for customizing your Tungsten Capture installation. This guide includes instructions for the following:

- Writing custom validation, recognition, and field scripts
- Creating and registering custom extensions such as custom modules, custom panels, workflow agents, and setup OCXs
- Using the Tungsten Capture API Library to create the custom extensions

Several custom script and extension samples are provided throughout this guide. This guide assumes that you have an understanding of the Tungsten Capture product and a working knowledge of an object-oriented development tool such as Visual Studio.

Related Documentation

In addition to this guide, the Tungsten Capture documentation set includes the following items.

Tungsten Capture Installation Guide

The *Tungsten Capture Installation Guide* contains essential information about installing Tungsten Capture and Tungsten Capture Network Server (Tungsten Capture Network).

This guide is for system administrators and developers who are installing Tungsten Capture or Tungsten Capture Network, or who need a description of the installation procedures and requirements.

Tungsten Capture Administrator's Guide

The *Tungsten Capture Administrator's Guide* contains essential information about configuring Tungsten Capture and Tungsten Capture Network Server (Tungsten Capture Network).

Tungsten Capture Help

Tungsten Capture online Help provides online assistance for system administrators and operators.

Tungsten Capture API References

The *Tungsten Capture API Reference* (APIRef.chm) is an online guide that provides the details of each API library needed to customize Tungsten Capture. The *Tungsten Capture Export Type Library API Reference* (APIRefExport.chm) gives details needed to customize an export connector. Both *API References* are intended to be used alongside the *Tungsten Capture Developer's Guide* as a primary resource for customizing Tungsten Capture.

You can access both *API References* in the following folder, which is available on your Tungsten Capture installation media:

Documentation\Help\APIRef

Tungsten Capture Release Notes

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

Training

Tungsten Automation offers both on-demand and instructor-led training to help you make the most of your product. To learn more about training courses and schedules, visit the [Tungsten Automation Learning Cloud](#).

Getting help with Tungsten Automation products

The [Tungsten Automation Knowledge Portal](#) repository contains articles that are updated on a regular basis to keep you informed about Tungsten Automation products. We encourage you to use the Knowledge Portal to obtain answers to your product questions.

To access the Tungsten Automation Knowledge Portal, go to <https://knowledge.tungstenautomation.com/>.

 The Tungsten Automation Knowledge Portal is optimized for use with Google Chrome, Mozilla Firefox, or Microsoft Edge.

The Tungsten Automation Knowledge Portal provides:

- Powerful search capabilities to help you quickly locate the information you need.
Type your search terms or phrase into the **Search** box, and then click the search icon.
- Product information, configuration details and documentation, including release news.
To locate articles, go to the Knowledge Portal home page and select the applicable Solution Family for your product, or click the View All Products button.

From the Knowledge Portal home page, you can:

- Access the Tungsten Automation Community (for all customers).
On the Resources menu, click the **Community** link.
- Access the Tungsten Automation Customer Portal (for eligible customers).
Go to the [Support Portal Information](#) page and click **Log in to the Customer Portal**.
- Access the Tungsten Automation Partner Portal (for eligible partners).
Go to the [Support Portal Information](#) page and click **Log in to the Partner Portal**.

- Access Tungsten Automation support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.

Go to the [Support Details](#) page and select the appropriate article.

Chapter 1

Overview

The *Tungsten Capture Developer's Guide* contains information about customizing your Tungsten Capture installation, including the following:

- Reasons to customize Tungsten Capture
- How to customize Tungsten Capture
- How to implement your custom Tungsten Capture components

This guide assumes that you have a good understanding of Tungsten Capture, Visual Basic or Visual Studio, and COM interface tools.

Reasons to Customize Tungsten Capture

Tungsten Capture provides the solution to your business needs for converting forms and data to a more useful electronic medium for data processing and archiving. However, you may also want to customize a Tungsten Capture process for a specific business task.

Some reasons to customize Tungsten Capture:

- To streamline or bypass unnecessary processes. For example, there may be no need to validate index data if the accuracy of the data exceeds a specified accuracy threshold defined by your business policy.
- To assert additional processes.
- To customize the user interface for your business environment. For example, a validation operator would need access only to certain functions or processes.

Because Tungsten Capture is modular (composed of the Administration, Scan, Validation, Verification, Quality Control, Export and other modules) you can customize a processing module using the Tungsten Capture API libraries provided with your product.

How to Customize Tungsten Capture

To create and write custom applications and scripts for Tungsten Capture, you can use tools such as VB.NET, C++, or C#, which are included in all Visual Studio editions (Community, Express, Professional, Ultimate, and Premium). You can use Visual Studio and VB.NET for creating custom scripts.

API libraries, which are documented in the *Tungsten Capture API Reference* and the *Tungsten Capture Export Type Library API Reference*, are available for you to create custom Tungsten Capture extensions.

You can access both *API References* in the following folder, which is available on your Tungsten Capture installation media:

Documentation\Help\APIRef

API Libraries

Tungsten Capture includes several API libraries that you can use to customize Tungsten Capture features or behavior. Details on these libraries are provided in the *Tungsten Capture API Reference*. Details about the Tungsten Capture Export Type Library are available separately, as noted in the table.

The Library Name column in the following table shows the library as it will be displayed in the Visual Basic References window. The Object and File Names column shows the library as it will be displayed in the Visual Basic Object Browser, followed by the name of the library file in parentheses.

The VB6 API libraries are deprecated and no longer used to develop new product features. For information about compatibility with existing VB6 APIs used in earlier versions of Kofax Capture, see [Backward Compatibility](#).

Library Name	Object and File Name	Usage
Tungsten Capture Module .NET Type Library	Kofax.Capture.CaptureModule (Kofax.Capture.CaptureModule.dll)	Used to add custom panels, menu items, and import applications to the Scan, Quality Control, Validation, and / or Verification modules.
Tungsten Capture Administration Module .NET Type Library	Kofax.Capture.AdminModule (Kofax.Capture.AdminModule.dll)	Used to create custom panels for the Administration module or Setup OCXs for custom modules or workflow agents.
Tungsten Capture Custom Module .NET Type Interface Library	Kofax.Capture.SDK.CustomModule (Kofax.Capture.SDK.CustomModule.dll)	Provides .NET type interfaces used to integrate a custom module into your Tungsten Capture installation. Implementation of this interface library is used with the Tungsten Capture Document Access .NET Type Implementation. Also referred to as the DBLite .NET Interface Library.

Library Name	Object and File Name	Usage
Tungsten Capture Document Access .NET Type Implementation Library	Kofax.Capture.DBLite (Kofax.DBLite.dll)	Used to integrate a custom module into your Tungsten Capture installation. Tungsten Capture Document Access .NET Type Implementation makes it possible for the custom module to access batch information from Tungsten Capture. This library also allows your custom module to relay batch information from the custom module to Tungsten Capture. Also referred to as the DBLite .NET Implementation Library.
Tungsten Capture Optimized Custom Module .NET Type Interface Library	Kofax.Capture.SDK.Data (Kofax.Capture.SDK.Data.dll)	Provides .NET type interfaces used for fast retrieval and selective update of data. Implementation of this interface library is used with the Tungsten Capture Optimized Custom Module .NET Type Implementation Library. Also referred to as the DbLiteOpt .NET Interface Library.
Tungsten Capture Optimized Custom Module .NET Type Implementation Library	Kofax.Capture.DBLiteOpt (Kofax.DBLiteOpt.dll)	An easy-to-use custom module interface used for fast retrieval and selective update of data. This API provides a mechanism that allows use of the Tungsten Capture Document Access .NET Type Implementation Library dynamic-link library (Kofax.DBLite.dll) to select and open a batch. Also referred to as the DBLiteOpt .NET Implementation Library.
Tungsten Capture Custom Workflow .NET Type Interface Library	Kofax.Capture.SDK.Workflow (Kofax.Capture.SDK.Workflow.dll)	Provides interfaces used by workflow agents to close batches by modules. Implementation of this interface library is used with the Tungsten Capture Custom Workflow .NET Type Implementation Library. Also referred to as the Workflow .NET Interface Library.

Library Name	Object and File Name	Usage
Tungsten Capture Custom Workflow .NET Type Implementation Library	Kofax.Capture.ACWFlib (Kofax.Capture.ACWFlib.dll)	An implementation of Tungsten Capture Custom Workflow .NET Type Interface Library. Also referred to as the Workflow .NET Implementation Library.
Tungsten Capture .NET Scripting Library	ScriptInterface (ScriptInterface.dll)	Used to create VB.NET custom field, validation, and recognition scripts.
Tungsten Capture Export Type Library	AscentRelease (AscRel.dll, Kofax.ReleaseLib.Interop.dll)	Used to create custom export connectors. Written in C++ and documented separately in APIRefExport.chm, which is available on your Tungsten Capture installation media in the following location: <code>Documentation\Help\APIRef</code>

Backward Compatibility

New features in Tungsten Capture do not use the VB6 public APIs, which are deprecated. However, your existing customizations based on VB6 public APIs are supported for use with Kofax Capture 11.1.0 or later, provided that you perform the installation without excluding VB6 components. If you specify the NoVB6 switch during the Tungsten Capture installation, or your organization policy does not allow installation of VB6 components, you cannot continue to use the customizations.

If you change your existing customizations based on existing VB6 public APIs, they should be recompiled against VB6 APIs from previous versions of Kofax Capture. Another option is to convert your project to Visual Studio 2015 or later, target the project to .NET 4.8, and then recompile using either the VB6 public APIs provided with Kofax Capture 11.1.0 or later, or the .NET APIs described in the *Kofax Capture API Reference*. The following table gives you information about specific versions.

If you need to refer to the VB6 API documentation, see the *API Reference* and *Developer's Guide* provided with the Kofax Capture 10.1 or 10.2 product.

Tool	Compiles Against VB6 and Interop Public APIs? Kofax Capture Version	Compiles Against .NET Public APIs? Kofax Capture Version	Description
Microsoft Visual Studio Express Community 2015, 2017 or 2019 Microsoft Visual Studio Enterprise 2015, 2017 or 2019 Microsoft Visual Studio Professional 2015, 2017 or 2019 Microsoft Visual Studio Express 2015 for Windows Desktop	Yes 8.0, 9.0, 10.0, 10.1, 10.2, 11.0, 11.1	Yes 10.1, 10.2, 11.0, 11.1	You can compile customizations against VB6/Interop Public APIs or .NET APIs, because both are available and valid with these Visual Studio versions.
Microsoft Visual Studio Community 2013 Microsoft Visual Studio Ultimate 2012 and 2013 Microsoft Visual Studio Premium 2012 and 2013 Microsoft Visual Studio Express 2012 and 2013 for Windows Desktop Microsoft Visual Studio Professional 2012 and 2013	Yes 8.0, 9.0, 10.0, 10.1, 10.2, 11.0, 11.1	Yes 10.1, 10.2, 11.0, 11.1	You can compile customizations against VB6/Interop Public APIs or .NET APIs, because both are available and valid with these Visual Studio versions. For Kofax Capture 11.1, it is necessary to install the Microsoft .NET Framework 4.8 Developer Pack and Language Packs, as these Visual Studio versions do not integrate with .NET Framework 4.8 by default.

Fluent User Interface (Fluent UI) Customizations

Some Tungsten Capture modules use the Fluent UI, similar to the user interface in recent versions of Microsoft Office. The Fluent UI is characterized by the use of a Ribbon, tabs, groups, and commands. The Fluent UI also supports right-click (context) menus, which are unchanged from prior interface approaches.

See the Tungsten Capture Help for more information on the Tungsten Capture user interface elements.

The following table shows differences between legacy terminology and the Fluent UI.

Legacy and Context Menu	Fluent UI	Comments
menu bar	Ribbon	An application window has only one Ribbon.
menu bar text /menu	tab	The Ribbon consists of a set of tabs. The strMenuBarText parameters specify the label that appears on the tab.

Legacy and Context Menu	Fluent UI	Comments
	group	<p>A "group" is an organizing container for commands (menu items) that belong to a tab. A tab may consist of many groups.</p> <p>Groups perform a function similar to divider bars in menus, but there is no programmatic mapping from a menu based interface to groups.</p> <p>When using older (unmodified) customizations, a group called "Menu Items" is automatically created. However, in the AddMenuEx method, the strGroupName parameter specifies the text to be used for the group name.</p>
menu text/menu item	command	<p>Commands can have downward pointing arrows that reveal a window with additional related commands. For example, the Paste command may open to show several options for pasting content.</p> <p>strMenuText parameters specify the label that appears on the command button.</p>

To maintain backward compatibility, the names and behaviors and documentation of existing methods and other API and customization elements are unchanged.

For example, the description, behavior and syntax of the SelectMenuItem method are unchanged. With a tab on the Ribbon, it causes the command (menu item) with the specified resource ID to be triggered. In a context menu, it causes the menu item with the specified resource ID to be triggered.

As another example, the ShowMenu method shows or hides a tab on the Ribbon. In the context of the Fluent UI, the strMenuBarText parameter specifies the label on the tab. The command label is determined by the strMenuText parameter.

Note that unless a method or other API element is specifically restricted to context menus, its documented functionality may apply to either the Fluent UI or a context menu (depending on the context of the application code).

Several recently added methods directly support the capabilities of the Fluent UI. See the *Tungsten Capture API Reference* for more information. We strongly recommend that you employ the newer methods to create or update your customizations.

Additional Resources

Files that define the data layout used when accessing Tungsten Capture data through custom modules and workflow agents are provided. Both mechanisms utilize identical data layouts.

These data layout files (AcBatch.htm, AcDocs.htm, and AcSetup.htm) are installed with Tungsten Capture in your Tungsten Capture installation folder.

The data layout is split into two files: AcSetup.htm defines setup information and AcBatch.htm defines runtime information. The data is partitioned into elements. Each element describes a specific kind of object and includes attributes and subelements. Attributes are properties that describe an element.

The file `AcDocs.htm` is a subset of `AcBatch.htm` and contains only the portion of `AcBatch` that relates to the document structure.

i Prior to Tungsten Capture 9, the product was named *Ascent Capture*. Although the product name in most of the documentation has changed, some instances of the prior product name appear in code snippets, file names, sample applications, API references, or references to prior versions of Tungsten Capture.

How to Implement Your Custom Scripts and Modules

Each chapter in this guide provides instructions and implementation details for creating custom extensions, such as custom modules, panels, workflow agents, and setup OCXs.

i Tungsten Capture 11.1.0 or later is based on Microsoft .NET Framework 4.8. To use Tungsten Capture 11.1.0 or later successfully with custom modules, export connectors, OCX/custom panels, or workflow agents based on .NET Framework versions earlier than 4.0, a special-purpose configuration file is required. For details, see the *Tungsten Capture Installation Guide*.

Chapter 2

Custom Script Creation Using VB.NET

Scripts are small programs used to perform specific tasks for associated Tungsten Capture modules. In the Administration module, you can set preferences for custom script creation.

This chapter explains how to create custom scripts using VB.NET as the script language and Visual Studio Express 2015 for Windows Desktop as the script editor. The advantage of writing scripts in VB.NET and the Microsoft Visual Studio development environment is support for Unicode, which is essential for supporting multi-byte character sets. Also, a large knowledge base for the Visual Studio development environment is available should you need additional coding assistance.

You should be familiar with programming concepts and the VB.NET programming language and development environment for writing custom scripts. You can create the following types of custom scripts in VB.NET:

- **Validation scripts** validate data in the Tungsten Capture Validation and Verification modules. For example, you can write a validation script that queries a database to verify that data for an index field matches the entered data. Document and folder validation scripts can be used to perform validation on document class index fields and folder index fields, respectively.
- **Recognition scripts** validate or modify data on results from the Recognition Server module. For example, you can write a recognition script that retrieves zone snippets from each image in a batch and determines if the zone meets a specific acceptance criteria.
- **Field scripts** validate data in index fields. For example, your field script can validate that data meets the criteria for a particular field type.

Software Requirements

The server and client workstations for Tungsten Capture must meet the system requirements listed on the Tungsten Automation Web site at <https://www.tungstenautomation.com/support>.

Also, when creating a custom script in VB.NET, one of the following development environments must be installed on your computer:

- Microsoft Visual Studio Enterprise 2015, 2017 or 2019
- Microsoft Visual Studio Professional 2015, 2017 or 2019
- Microsoft Visual Studio Community 2013, 2015, 2017 or 2019
- Microsoft Visual Studio Express 2012, 2013 or 2015 for Windows Desktop
- Microsoft Visual Studio Ultimate 2012 or 2013
- Microsoft Visual Studio Premium 2012 or 2013
- Microsoft Visual Studio Professional 2012 or 2013

You also need Microsoft .NET Framework 4.8 runtime installed in your development environment. If not already installed, the .NET 4.8 runtime is installed either by Tungsten Capture or by Visual Studio 2019.

Also, to compile against Tungsten Capture 10.2 or 10.1 libraries that are targeted to .NET 4.0, you must use Microsoft Visual Studio 2010. You can use Visual Studio 2008 to compile Tungsten Capture 10.0, 9.0 and 8.0. See [Backward Compatibility](#) for more information.

 Visual Studio Express and Visual Studio Community are free downloads from the Microsoft Web site.

Validation Script Creation in VB.NET

Validation scripts are useful for verifying that data meets specific formatting criteria or for validating database information for fields of a document class. You can perform these checks before and after document processing (that is, DocumentPreProcessing and DocumentPostProcessing events).

In this section, you will learn about:

- How to select the kind of custom script to create using VB.NET
- Objects, methods, and properties that are available for use from the Tungsten Capture .NET Scripting API
- VB.NET project location for your script
- Deployment of the script's Visual Basic project
- Publishing requirements for the script

A sample validation script written in VB.NET is provided.

Selecting the Scripting Language

You can create a custom script from the Tungsten Capture Administration module. A document class must exist before you create a validation script.

1. On the **Home** tab, in the **Document Class** group, click **Validation Script**.
The **Validation Script** window appears.
2. On the **Document classes** list, select the document class for which you want to create a script.
3. On the **Scripting language** list, select **VB.NET**.
4. Click **Create**.

A VB.NET project is created for the script, and it opens in the code editor of your .NET application. Add your code to create the custom script.

Tungsten Capture .NET Scripting API

Your document validation script project has access to the events and properties of the Tungsten Capture .NET Scripting library. Each script can consist of several events and event handlers. The index fields that exist for the document class selected are included in the script code shell.

DocumentValidationScript Class

The DocumentValidationScript class contains the events that are available for use in a validation script. You add code for a selected event to perform a custom data validation routine. The following events, event arguments, and their associated properties can be used.

Validation Script Events

Events	Description
BatchLoading	Called when a batch is first opened in the Validation or Verification module. If a batch has multiple document classes, the function is called once per document class the first time a document class is processed.
BatchUnloading	Called when a batch is closed.
DocumentPreProcessing	Called each time a new document is opened.
DocumentPostProcessing	Called after each document is closed.
PreDocumentEventArgs	This class represents the event arguments for the DocumentValidationScript.DocumentPreProcessing event.
PostDocumentEventArgs	This class represents the event arguments for the DocumentValidationScript.DocumentPostProcessing event.

Exceptions

To signal an error state, the VB.NET script can throw an exception during event handling. Three types of exceptions are available:

- FatalErrorException
- RejectAndSkipDocumentException
- ValidationErrorException

For details about each event, event arguments, and properties, refer to the *Tungsten Capture API Reference*. For more information about exceptions, refer to [Error Handling in VB.NET](#).

VB.NET Project File Location

The file name assigned to the script is shown in the title bar of the programming product. The default location for each project is a numeric folder name under the `~AdminDB\Scripts` folder.

Default location of the AdminDB folder:

- **Server:** \\<server name>\CaptureSV
- **Standalone:** C:\ProgramData\Tungsten Automation\Capture

i Validation of a batch field and batch totals through VB.NET scripting is not supported. However, batch fields are exposed through the Batch object, which can be accessed from the parameter of the event handler.

You must compile the script and publish your batch class before the script can be used in batches. Your script cannot be applied to a batch created before the new publication date. In addition, you must republish if you make changes to the script.

i When a validation script is created, the script is stored in a folder named with the document class name, which becomes part of the final script folder path. Folder paths cannot exceed 256 characters. Therefore, ensure that your document class names are not excessively long.

Deployment of a VB.NET Project

VB.NET scripts have a folder of source files and a folder of executables. The entire VB.NET script project is deployed to the `Local\Scripts` folder before the Validation or Recognition Server module is launched for either a standalone or remote/central site environment.

The VB.NET script is opened each time a batch is opened (if the script is not already present) and a new script ID folder is created for a published batch class. VB.NET scripts can include field scripts, which are executed when there is no document/folder validation script.

Batches using VB.NET scripts are deployed automatically on standalone workstations and on Tungsten Capture Network Server remote sites through synchronization by the Remote Synchronization Agent. Scripts are downloaded when the remote site synchronizes with the central site.

Script Publishing Requirements

A VB.NET script must be compiled before it can be published. Otherwise, an error occurs.

The publish check is performed only on a newly created VB.NET script. The publish check is not performed on updated or changed scripts, and it is the responsibility of the script developer to recompile scripts as needed. Also, a publish check is not performed on VB.NET scripts for imported and exported batch classes.

Sample VB.NET Validation Script

This section describes how to create a sample VB.NET document validation script named `SampleScript`, which contains the fields `ClientName` and `SSN`. The sample script is used in the next section, which explains how to add custom code to validate a social security number.

i To use the sample, you need a database, which is used for the SSN value lookup. You will need the database name, password, and host name for the location where the database resides.

1. Navigate to the Tungsten Capture program folder and click **Administration**.
2. Create a document class with the name **SampleScript**.
3. On the Document class tree view, select **SampleScript**, right-click, and then click **Properties**. The Document Class Properties window appears.
4. On the General tab, use **New Index Field** to add two fields with the following information:
 - a. **Name:** `ClientName`, **Field Type:** `Alphanumeric_25`, **Default** set to empty, **Required** set to `True`.

- b. **Name:** SSN, **Field Type:** Alphanumeric_12, **Default** set as empty, **Required** set to True.
 - c. Save the changes and close the **Document Class Properties** window.
5. Verify that the SampleScript document class is selected.
6. On the **Home** tab, in the **Document Class** group, click **Validation Script**.
7. On the **Document Validation Script** window, on the **Document** classes list, select **SampleScript**.
8. On the Scripting language list, select **VB.NET**.
9. Click **Create**.

Sample Validation Script Code

Using the sample validation script created in the previous section, you can add code that compares and validates a social security number field from a scanned document to a social security number that is already in the database for the client. The first function of the code is the FieldPostProcessing event followed by helper functions.

After you add the following code, compile the script. When the script is successfully compiled, publish the batch class by right-clicking the batch class and selecting **Publish**.

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports Kofax.AscentCapture.NetScripting
Imports Kofax.Capture.CaptureModule.InteropServices
Imports Kofax.Capture.CaptureModule
Imports System.Data.SqlClient

Namespace SampleScript
  <SuppressFieldEventsOnDocClose(False)> _
  Public Class SampleScript
    Inherits DocumentValidationScript

    <IndexFieldVariableAttribute("ClientName")> _
    Dim WithEvents ClientName As FieldScript

    <IndexFieldVariableAttribute("SSN")> _
    Dim WithEvents SSN As FieldScript

    Private Sub SSN_FieldPostProcessing(ByVal sender As Object, ByVal e As
Kofax.AscentCapture.NetScripting.PostFieldEventArgs) Handles SSN.FieldPostProcessing
      Dim oConnection As IDbConnection
      oConnection = OpenDatabaseConnection()
      Try
        oConnection.Open()
        If (Not FindSsnInDatabase(oConnection, SSN.IndexField.Value)) Then
          Throw New Kofax.AscentCapture.NetScripting.ValidationErrorException("SSN missing
from the database.", SSN.IndexField)
        End If
      Finally
        oConnection.Close()
      End Try
    End Sub

    Private Function OpenDatabaseConnection() As IDbConnection
      Dim oConnectionStringBuilder As SqlConnectionStringBuilder
      oConnectionStringBuilder = New SqlConnectionStringBuilder()
      oConnectionStringBuilder.DataSource = "DBServer00\SQL2005" _
```

```

'*** server name
oConnectionStringBuilder.UserID = "User1" '*** user name
oConnectionStringBuilder.Password = "abc123" '*** password
oConnectionStringBuilder.InitialCatalog = "SocialSecurityDB" _
'*** database name
Dim oConnection As SqlConnection
oConnection = New SqlConnection
oConnection.ConnectionString = oConnectionStringBuilder.ConnectionString
Return oConnection
End Function

Private Function FindSsnInDatabase(ByVal oConnection As IDbConnection, _
ByVal strSsn As String) As Boolean
Dim oCommand As IDbCommand
oCommand = oConnection.CreateCommand()
'*** The database has a "Customers" table with a column for SSN
oCommand.CommandText = String.Format("SELECT SSN FROM Customers WHERE SSN=N'{0}'",
strSsn)
oCommand.CommandType = CommandType.Text
Dim oReader As IDataReader
oReader = oCommand.ExecuteReader()
Dim bFound As Boolean
bFound = oReader.Read()
oReader.Close()
Return bFound
End Function
End Class
End Namespace

```

Testing VB.NET Custom Scripts

You can test and debug a VB.NET custom validation script without republishing the associated batch class. However, the batch class (with one or more associated VB.NET scripts) must have been previously published and a batch must be ready to be processed.

When a VB.NET script is tested, the script in the Administration module is loaded instead of the published script. In this case, the VB.NET script being debugged is the unpublished copy. However, the batch class and script must be republished before changes can take effect.

 The ability to debug and modify the VB.NET script without republishing the batch class is not supported in Visual Basic Express Edition. It is only supported in Visual Studio.

When debugging a validation script in VB.NET, if you stop the debugger while a batch is open, the validation process is terminated and the batch is left in an "In progress" state. It may take time for the batch to return to the "Ready" state.

Here is the typical process for testing and debugging a VB.NET document validation script:

1. A user with administrator rights creates a VB.NET document validation script in the Administration module and publishes a batch class.
2. A batch is created and processed through the Validation module.
3. The VB.NET script project is opened in Visual Studio.
4. Break points are set in the script, and changes are made to the script.
5. The validation script is run from Visual Studio, and the debugger stops at the first break point.

6. Testing and debugging continues until the script is ready to use.
7. The script is compiled, and the batch is published.

Error Handling in VB.NET

A VB.NET script uses an exception during event handling to signal an error state. The Tungsten Capture .NET Scripting API provides the following exceptions:

- FatalErrorException
- RejectAndSkipDocumentException
- ValidationErrorException

FatalErrorException

Used to signal a fatal error, this exception can be used in validation, recognition, and field scripts. When this exception is thrown, the error message is displayed in Validation or Verification, or logged in the Recognition Server. The batch is set to an error state and sent to Quality Control.

Example:

```
Throw New FatalErrorException("Missing validation resource")
```

RejectAndSkipDocumentException

Used by the validation script to reject and skip a document and to advance to the next document, this exception is thrown in document validation scripts only.

The FieldScript.FieldPreProcessing and FieldScript.FieldPostProcessing events for any unvalidated fields in the document are skipped. However, the DocumentValidationScript.DocumentPostProcessing event is called.

Example:

```
Throw New FatalErrorException("Missing validation resource")
```

ValidationErrorException

Used to signal a validation error, this exception is to be thrown in validation scripts only. If the Recognition Server receives this exception, It is treated as a fatal error.

There are two versions of this exception:

- ValidationErrorException (ErrMsg)
- ValidationErrorException (ErrMsg, IndexField)

ValidationErrorException (ErrMsg)

If this version is used, the error message is displayed in the status bar of the Validation or Verification module and will not advance the focus to the next field. The focus remains on the last field that is being validated.

Example:

```
Throw New ValidationErrorException()
```

ValidationErrorException (ErrMsg, IndexField)

This version is similar to the previous version for ValidationErrorException, except that it allows the developer to set the focus on a particular field. IndexField specifies the field to get the focus. If the field does not exist or is null, then the last field being evaluated gets the focus.

For example, if cross-field validation is being performed in the DocumentValidationScript.DocumentPostProcessing event and a computation routine determines that the third field does not match the sum of the first and second fields, an exception is thrown with the third field getting the focus.

Example:

```
Throw New ValidationErrorException("Sum does not match",oIndexField)
```

 If ErrorMsg is empty, then the default error message is displayed.

Refer to the *Tungsten Capture API Reference* for details about the syntax and parameters for these exception classes.

Recognition Script Creation in VB.NET

Recognition scripts are useful for processing zone snippets with a custom recognition engine, modifying the process of a Tungsten Automation recognition engine, or performing an offline recognition process.

In this section, you will learn about the following:

- How to select VB.NET as the scripting language for the recognition script.
- Objects, methods, and properties that are available for use from the Tungsten Capture .NET Scripting API.

The VB.NET project location of your script, the script deployment, and publishing requirements are the same as those for validation scripts.

Tungsten Capture .NET Scripting API

Your recognition script project has access to the events and properties of the Tungsten Capture .NET Scripting library. Each script can consist of several events and event handlers.

RecognitionScript Class

The RecognitionScript class contains the events that are available for use in a recognition script. You add code for a selected event to perform a custom recognition routine. You can use the following events, event arguments, and their associated properties.

Recognition Script Events

Events	Description
BatchLoading	Called when a batch is opened.
BatchUnloading	Called when a batch is closed.
RecognitionPreProcessing	Called before each zone snippet is processed.
RecognitionPostProcessing	Called after each zone snippet is processed.
PreRecognitionEventArgs	This class represents the event arguments for the RecognitionScript.RecognitionPreProcessing event.
PostRecognitionEventArgs	This class represents the event arguments for the RecognitionScript.RecognitionPostProcessing event.

Exceptions

To signal an error state, the VB.NET script can throw an exception during event handling. Three types of exceptions are available:

- FatalErrorException
- RejectAndSkipDocumentException
- ValidationErrorException

For details about each event, event arguments, and properties, refer to the *Tungsten Capture API Reference*. For more information about exceptions, refer to [Error Handling in VB.NET](#).

Creating a Recognition Script Using VB.NET

You can use the Tungsten Capture Administration module to create a recognition script. VB.NET recognition scripts can be created for separator zones, form identification zones, or index zones.

 You can create scripts only for custom recognition profiles. You cannot create scripts for full text OCR or page-level form identification profiles.

1. On the **Tools** tab, in the **Recognition** group, click **Scripts**.
The **Recognition Script** window appears.
2. On the **Recognition profiles** list, select the custom recognition profile for which you want to create a script.
3. In the **Scripting language** box, select the scripting language. In this case, select **VB.NET**.
4. Click **Create**.

A new script is created and displayed in the VB.NET code editor. The file name assigned to the script appears in the title bar.

5. When finished writing your code, compile the script, exit the editor, and close the **Recognition Script** window.

You must compile the script and publish your batch class before the script can be used in batches. Your script cannot be applied to a batch created before the new publication date. In addition, you must republish if you make changes to the script.

Debugging Your Settings with a VB.NET Recognition Script

You can use a VB.NET recognition script to display the zone snippet when the batch is processed in the Recognition Server module. This is a good way to test your settings.

1. In the Administration module, create a recognition profile for a zone.
2. On the **Tools** tab, in the **Recognition** group, click **Scripts**.
The **Recognition Script** window appears.
3. Select a profile from the list of **Recognition** profiles.
4. Select a scripting language from the list of Scripting languages. In this case, select **VB.NET**.
5. Select **Create**.

A new script appears in the VB.NET code editor.

Sample Code

Publish the batch class and process a test batch through the Recognition Server module. For every zone to which the recognition profile is attached, a message displays the file name of the zone snippet, the value, and the confidence. If you have access to a viewer, you can display the zone snippet file to see the zone snippet image.

```
Private Sub CustomRecognitionProfile_BatchLoading(ByVal sender As Object, ByRef
    ImageFileRequired As Boolean) Handles Me.BatchLoading
    '*** Enable the image file property when loading the batch.
    ImageFileRequired = True
End Sub
Private Sub CustomRecognitionProfile_RecognitionPostProcessing(ByVal _
    sender As Object, _
    ByVal e As Kofax.AscentCapture.NetScripting.PostRecognitionEventArgs) Handles
    Me.RecognitionPostProcessing
    '*** Display a message box with the image file, value, and confidence
    '*** for each field.
    MessageBox.Show(e.ImageFile + " " + e.Value + " " + e.Confidence.ToString())
End Sub
```

Sample VB.NET Recognition Script

The following recognition script demonstrates how to set recognition criteria to enhance a field that may display poorly. The confidence level is set to 75%. If the confidence level for the field falls below 75%, the document is sent to the Quality Control module.

```
Private Sub ConfidenceScript_RecognitionPostProcessing( _
    ByVal sender As Object, ByVal e As
    Kofax.AscentCapture.NetScripting.PostRecognitionEventArgs) _
    Handles Me.RecognitionPostProcessing
    If (e.Confidence < 75) Then
        Throw New Kofax.AscentCapture.NetScripting.FatalErrorException _
            ("Confidence value was less than 75 percent.")
    End If
End Sub
```

```
End If
End Sub
```

Removing a Recognition Script

You can remove a recognition script to restore the default recognition procedures.

Use the Administration module to remove recognition scripts. When you remove a recognition script, the source code file is not deleted from the Scripts folder. Although you cannot select the file to "reattach" it to a recognition profile, you can add the customizations to a new script. The easiest way to do this is to create a new recognition script and open the old recognition script in a text editor such as Notepad. Then, you can copy code from the old script to the new script.

1. On the **Tools** tab, in the **Recognition** group, click **Scripts**.
The **Recognition Script** window appears.
2. On the **Recognition** profiles list, select the recognition profile for which you want to restore the default processing.
3. Click **Remove**.
The script is no longer associated with the recognition profile.

Field script

A field script is a small program that can contain variables and functions needed for validating an index field with an associated field type.

A field script is invoked when there is no document validation script for a document class, and the field script can be referenced by other custom scripts for documents. A field script can have only one script language defined.

A field script can contain field formatting functions and event arguments associated with each event.

Field Script Events

Event	Description
FieldFormatting	Called for every field as every other field is exited. Allows the display of a field to differ from the stored value.
FieldPreProcessing	Called as the field is entered. Set e.SkipMode = SaveAndSkipDocumentOrFolder to save the field and move to the next document or folder. Set e.SkipMode = SaveAndSkipField to save the field and move to the next field.
FieldPostProcessing	Called as the field is exited. Set e.SaveAndSkipDocument = true to save the field and move to the next document.
FormatFieldEventArgs	This class represents the event arguments for the FieldScript.FieldFormatting event.
PreFieldEventArgs	This class represents the event arguments for the FieldScript.FieldPreProcessing event.
PostFieldEventArgs	This class represents the event arguments for the FieldScript.FieldPostProcessing event.

For details about each event, event arguments, and properties, refer to the *Tungsten Capture API Reference*.

Sample VB.NET Field Script

When you create a new field script, a new class is derived from the `FieldScript` class of the Tungsten Capture NET Scripting API. The name of the class is based on the field type name (all non-alphanumeric characters of the class name are replaced with underscore characters). The name of the class can be changed; however, the changed name is not automatically updated in the generated field script.

The following is a field script sample that changes all mixed case letters of a particular field to upper case letters for reading clarity.

```
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports Kofax.AscentCapture.NetScripting
Imports Kofax.Capture.CaptureModule.InteropServices

Namespace UppercaseField20

    Public Class UppercaseField20
        Inherits FieldScript

        Private Sub UppercaseField20_FieldFormatting(ByVal sender As Object, ByVal e As
            Kofax.AscentCapture.NetScripting.FormatFieldEventArgs) Handles Me.FieldFormatting
            '*** Convert the string to upper case.
            Dim strUpperCase As String
            strUpperCase = Me.IndexField.Value.ToUpper()
            '*** Set the value to the uppercase string.
            Me.IndexField.Value = strUpperCase
        End Sub
    End Class
End Namespace
```

Chapter 3

Custom Script Creation Using Visual C#

Scripts are small programs used to perform specific tasks for associated Tungsten Capture modules. In the Administration module, you can set preferences for custom script creation.

This chapter explains how to create custom scripts using C# as the script language and Visual Studio Express 2015 for Windows Desktop as the script editor. The advantage of writing scripts in Visual C# and the Microsoft Visual Studio development environment is support for Unicode, which is essential for supporting multi-byte character sets. Also, a large knowledge base for the Visual Studio development environment is available should you need additional coding assistance.

You should be familiar with programming concepts and the Visual C# programming language and development environment for writing custom scripts. You can create the following types of custom scripts in Visual C#:

- **Validation scripts** validate data in the Tungsten Capture Validation and Verification modules. For example, you can write a validation script that queries a database to verify that data for an index field matches the entered data. Document and folder validation scripts can be used to perform validation on document class index fields and folder index fields, respectively.
- **Recognition scripts** validate or modify data on results from the Recognition Server module. For example, you can write a recognition script that retrieves zone snippets from each image in a batch and determines if the zone meets a specific acceptance criteria.
- **Field scripts** validate data in index fields. For example, your field script can validate that data meets the criteria for a particular field type.

Software Requirements

The server and client workstations for Tungsten Capture must meet the system requirements listed on the Tungsten Automation Web site at <https://www.tungstenautomation.com/support>.

Also, when creating a custom script in C#, one of the following development environments must be installed on your computer:

- Microsoft Visual Studio Enterprise 2015 or 2017
- Microsoft Visual Studio Professional 2015 or 2017
- Microsoft Visual Studio Community 2013, 2015, or 2017
- Microsoft Visual Studio Enterprise 2015 or 2017
- Microsoft Visual Studio Express 2012, 2013 or 2015 for Windows Desktop
- Microsoft Visual Studio Ultimate 2012 or 2013
- Microsoft Visual Studio Premium 2012 or 2013
- Microsoft Visual Studio Professional 2012 or 2013

You also need Microsoft .NET Framework 4.6.1 runtime installed in your development environment. If not already installed, the .NET 4.6.1 runtime is installed either by Tungsten Capture or by Visual Studio 2015 or 2017.

Also, to compile against Tungsten Capture 10.2 or 10.1 libraries that are targeted to .NET 4.0, you must use Microsoft Visual Studio 2010. You can use Visual Studio 2008 to compile Tungsten Capture 10.0, 9.0 and 8.0. See [Backward Compatibility](#) for more information.

 Visual Studio Express and Visual Studio Community are free downloads from the Microsoft Web site.

Validation Script Creation in Visual C#

Validation scripts are useful for verifying that data meets specific formatting criteria or for validating database information for fields of a document class. You can perform these checks before and after document processing (that is, DocumentPreProcessing and DocumentPostProcessing events).

In this section, you will learn about:

- How to select the kind of custom script to create using Visual C#
- Objects, methods, and properties that are available for use from the Tungsten Capture .NET Scripting API
- Visual C# project location for your script
- Deployment of the script's Visual C# project
- Publishing requirements for the script

A sample validation script written in Visual C# is provided.

Selecting the Scripting Language

Although it is possible to create a custom script outside Tungsten Capture using a supported Visual Studio environment, you typically create a custom script from the Tungsten Capture Administration module.

1. On the **Home** tab, in the **Document Class** group, click **Validation Script**.
The **Validation Script** window appears.
2. On the **Document classes** list, verify that the applicable document class is selected.
3. On the **Scripting language** list, select **Visual C#**.
4. In the **Script name** box, assign a name to the validation script.
5. Click **Create**.
A Visual C# project is created for the script, and Visual Studio is opened.
6. In Visual Studio, open the .cs file, which initially looks similar to the following sample.

```
using Kofax.AscentCapture.NetScripting;  
using Kofax.Capture.CaptureModule.InteropServices;  
using System;  
using System.Collections.Generic;  
using System.Text;
```

```

namespace TestDoc {

    [SuppressFieldEventsOnDocClose(false)]
    public class TestDoc : DocumentValidationScript {

        [IndexFieldVariableAttribute("Name0")]
        FieldScript Name0;

        public TestDoc(bool bIsValidation, string strUserID, string strLocaleName)
        : base(bIsValidation, strUserID, strLocaleName)
        {}
    }
}

```

7. Continue to Customize the Script.

Tungsten Capture Visual C# Scripting API

Your document validation script project has access to the events and properties of the Tungsten Capture Visual C# Scripting library. Each script can consist of several events and event handlers. The index fields that exist for the document class selected are included in the script code shell.

DocumentValidationScript Class

The DocumentValidationScript class contains the events that are available for use in a validation script. You add code for a selected event to perform a custom data validation routine. The following events, event arguments, and their associated properties can be used.

Validation Script Events

Events	Description
BatchLoading	Called when a batch is first opened in the Validation or Verification module. If a batch has multiple document classes, the function is called once per document class the first time a document class is processed.
BatchUnloading	Called when a batch is closed.
DocumentPreProcessing	Called each time a new document is opened. Set e.SaveAndSkip = true to save the field and move to the next document.
DocumentPostProcessing	Called after each document is closed.
PreDocumentEventArgs	This class represents the event arguments for the DocumentValidationScript.DocumentPreProcessing event.
PostDocumentEventArgs	This class represents the event arguments for the DocumentValidationScript.DocumentPostProcessing event.

Exceptions

To signal an error state, the Visual C# script can throw an exception during event handling. Three types of exceptions are available:

- FatalErrorException
- RejectAndSkipDocumentException

- ValidationErrorException

For details about each event, event arguments, and properties, refer to the *Tungsten Capture API Reference*. For more information about exceptions, refer to [Error Handling in Visual C#](#).

Creating a Custom Visual C# Validation Script

This section describes how to create a custom Visual C# document validation script named TestDoc.

When creating a new script, you are provided with one method that is the constructor for the class, such as the “TestDoc” class in the following procedure. In this constructor, the BatchLoading and BatchUnloading events need to be subscribed to, and the corresponding event method implementations must be created.

1. Create the script as described in [Selecting the Scripting Language](#).
2. In the script project window in Visual Studio, in the constructor method, type “this.” and select BatchLoading.
3. Type “+=” and you will see a message such as:

```
TestDoc_BatchLoading (Press TAB to insert)
```
4. Press the Tab key to automatically complete the declaration, and another message appears:

```
Press TAB to generate handler 'TestDoc_BatchLoading' in this class
```
5. Press the Tab key again to automatically generate the BatchLoading event method in the class.
6. Repeat the same steps to generate the BatchUnloading event method.
7. The following exception code is automatically added to all event methods created in this manner. Remove this line:

```
throw new NotImplementedException();
```
8. Now, subscribe to the Document events. In the constructor of the class, type “this.” Implement the Document PreProcessing and PostProcessing event methods in the same manner described earlier.

i To have the DocumentPreProcessing and DocumentPostProcessing events fire on each document, you must add the event handlers in the constructor of the script class (and not in the BatchLoading event handler). This approach is similar to the one used with the AddHandler and RemoveHandler functions for VB.NET.

9. Subscribe and implement the Index Field PreProcessing, PostProcessing and FieldFormatting events, in the same manner described earlier. However, in this case, Field events need to be subscribed to in the BatchLoading event method. Also, the Index Field name is used in the declaration.
10. The following example shows the results for the constructor.

```
public TestDoc(bool bIsValidatation, string strUserID, string strLocaleName)
    : base(bIsValidatation, strUserID, strLocaleName)
{
    this.BatchLoading += TestDoc_BatchLoading;
    this.BatchUnloading += TestDoc_BatchUnloading;
    this.DocumentPreProcessing += TestDoc_DocumentPreProcessing;
    this.DocumentPostProcessing += TestDoc_DocumentPostProcessing;
}
```

11. The final step is to unsubscribe from the events in the BatchUnloading event method.

12. Verify that the generated event methods are similar to the following example:

```

void TestDoc_BatchLoading(object sender, BatchEventArgs e)
{
this.Name0.FieldPreProcessing += Name0_FieldPreProcessing;
this.Name0.FieldPostProcessing += Name0_FieldPostProcessing;
this.Name0.FieldFormatting += Name0_FieldFormatting;
}
void TestDoc_BatchUnloading(object sender, BatchEventArgs e)
{
Name0.FieldFormatting -= Name0_FieldFormatting;
Name0.FieldPostProcessing -= Name0_FieldPostProcessing;
Name0.FieldPreProcessing -= Name0_FieldPreProcessing;
DocumentPostProcessing -= TestDoc_DocumentPostProcessing;
DocumentPreProcessing -= TestDoc_DocumentPreProcessing;
BatchLoading -= TestDoc_BatchLoading;
BatchUnloading -= TestDoc_BatchUnloading;
}
void TestDoc_DocumentPreProcessing(object sender, PreDocumentEventArgs e)
{
}
void TestDoc_DocumentPostProcessing(object sender, PostDocumentEventArgs e)
{
}
void Name0_FieldPreProcessing(object sender, PreFieldEventArgs e)
{
}
void Name0_FieldPostProcessing(object sender, PostFieldEventArgs e)
{
}
void Name0_FieldFormatting(object sender, FormatFieldEventArgs e)
{
}
}

```

Visual C# Project File Location

The file name assigned to the script is shown in the title bar of the programming product. The default location for each project is a numeric folder name under the `~AdminDB\Scripts` folder.

Default location of the `AdminDB` folder:

- **Server:** \\<server name>\CaptureSV
- **Standalone:** C:\ProgramData\Tungsten Automation\Capture

i Validation of a batch field and batch totals through Visual C# scripting is not supported. However, batch fields are exposed through the `Batch` object, which can be accessed from the parameter of the event handler.

You must compile the script and publish your batch class before the script can be used in batches. Your script cannot be applied to a batch created before the new publication date. In addition, you must republish if you make changes to the script.

i When a validation script is created, the script is stored in a folder named with the document class name, which becomes part of the final script folder path. Folder paths cannot exceed 256 characters. Therefore, ensure that your document class names are not excessively long.

Deployment of a Visual C# Project

Visual C# scripts have a folder of source files and a folder of executables. The entire Visual C# script project is deployed to the `Local\Scripts` folder before the Validation or Recognition Server module is launched for either a standalone or remote/central site environment.

The Visual C# script is opened each time a batch is opened (if the script is not already present) and a new script ID folder is created for a published batch class. Visual C# scripts can include field scripts, which are executed when there is no document/folder validation script.

Batches using Visual C# scripts are deployed automatically on standalone workstations and on Tungsten Capture Network Server remote sites through synchronization by the Remote Synchronization Agent. Scripts are downloaded when the remote site synchronizes with the central site.

Script Publishing Requirements

A Visual C# script must be compiled before it can be published. Otherwise, an error occurs.

The publish check is performed only on a newly created Visual C# script. The publish check is not performed on updated or changed scripts, and it is the responsibility of the script developer to recompile scripts as needed. Also, a publish check is not performed on Visual C# scripts for imported and exported batch classes.

Testing Visual C# Custom Scripts

You can test and debug a Visual C# custom validation script without republishing the associated batch class. However, the batch class (with one or more associated Visual C# scripts) must have been previously published and a batch must be ready to be processed.

When a Visual C# script is tested, the script in the Administration module is loaded instead of the published script. In this case, the Visual C# script being debugged is the unpublished copy. However, the batch class and script must be republished before changes can take effect.

 The ability to debug and modify the Visual C# script without republishing the batch class is not supported in Visual C# Express Edition. It is only supported in Visual Studio.

When debugging a validation script in Visual C#, if you stop the debugger while a batch is open, the validation process is terminated and the batch is left in an "In progress" state. It may take time for the batch to return to the "Ready" state.

Here is the typical process for testing and debugging a Visual C# document validation script:

1. A user with administrator rights creates a Visual C# document validation script in the Administration module and publishes a batch class.
2. A batch is created and processed through the Validation module.
3. The Visual C# script project is opened in Visual Studio.
4. Break points are set in the script, and changes are made to the script.

5. The validation script is run from Visual Studio, and the debugger stops at the first break point.
6. Testing and debugging continues until the script is ready to use.
7. The script is compiled, and the batch is published.

Error Handling in Visual C#

A Visual C# script uses an exception during event handling to signal an error state. The Tungsten Capture .NET Scripting API provides the following exceptions:

- FatalErrorException
- RejectAndSkipDocumentException
- ValidationErrorException

FatalErrorException

Used to signal a fatal error, this exception can be used in validation, recognition, and field scripts. When this exception is thrown, the error message is displayed in Validation or Verification, or logged in the Recognition Server. The batch is set to an error state and sent to Quality Control.

Example:

```
throw new FatalErrorException("Missing validation resource")
```

RejectAndSkipDocumentException

Used by the validation script to reject and skip a document and to advance to the next document, this exception is thrown in document validation scripts only.

The FieldScript.FieldPreProcessing and FieldScript.FieldPostProcessing events for any unvalidated fields in the document are skipped. However, the DocumentValidationScript.DocumentPostProcessing event is called.

Example:

```
throw new FatalErrorException("Missing validation resource")
```

ValidationErrorException

Used to signal a validation error, this exception is to be thrown in validation scripts only. If the Recognition Server receives this exception, It is treated as a fatal error.

There are two versions of this exception:

- ValidationErrorException (ErrMsg)
- ValidationErrorException (ErrMsg, IndexField)

ValidationErrorException (ErrMsg)

If this version is used, the error message is displayed in the status bar of the Validation or Verification module and will not advance the focus to the next field. The focus remains on the last field that is being validated.

Example:

```
throw new ValidationErrorException()
```

ValidationErrorException (ErrMsg, IndexField)

This version is similar to the previous version for ValidationErrorException, except that it allows the developer to set the focus on a particular field. IndexField specifies the field to get the focus. If the field does not exist or is null, then the last field being evaluated gets the focus.

For example, if cross-field validation is being performed in the DocumentValidationScript.DocumentPostProcessing event and a computation routine determines that the third field does not match the sum of the first and second fields, an exception is thrown with the third field getting the focus.

Example:

```
throw new ValidationErrorException("Sum does not match", oIndexField)
```

 If ErrorMsg is empty, then the default error message is displayed.

Refer to the *Tungsten Capture API Reference* for details about the syntax and parameters for these exception classes.

Recognition Script Creation in Visual C#

Recognition scripts are useful for processing zone snippets with a custom recognition engine, modifying the process of a Tungsten Automation recognition engine, or performing an offline recognition process.

In this section, you will learn about the following:

- How to select Visual C# as the scripting language for the recognition script.
- Objects, methods, and properties that are available for use from the Tungsten Capture .NET Scripting API.

The Visual C# project location of your script, the script deployment, and publishing requirements are the same as those for validation scripts.

Tungsten Capture Visual C# Scripting API

Your recognition script project has access to the events and properties of the Tungsten Capture Visual C# Scripting library. Each script can consist of several events and event handlers.

RecognitionScript Class

The RecognitionScript class contains the events that are available for use in a recognition script. You add code for a selected event to perform a custom recognition routine. You can use the following events, event arguments, and their associated properties.

Recognition Script Events

Events	Description
BatchLoading	Called when a batch is opened.
BatchUnloading	Called when a batch is closed.
RecognitionPreProcessing	Called before each zone snippet is processed.
RecognitionPostProcessing	Called after each zone snippet is processed.
PreRecognitionEventArgs	This class represents the event arguments for the RecognitionScript.RecognitionPreProcessing event.
PostRecognitionEventArgs	This class represents the event arguments for the RecognitionScript.RecognitionPostProcessing event.

Exceptions

To signal an error state, the Visual C# script can throw an exception during event handling. Three types of exceptions are available:

- FatalErrorException
- RejectAndSkipDocumentException
- ValidationErrorException

For details about each event, event arguments, and properties, refer to the *Tungsten Capture API Reference*. For more information about exceptions, refer to [Error Handling in Visual C#](#).

Creating a Recognition Script Using Visual C#

You can use the Tungsten Capture Administration module to create a recognition script. Visual C# recognition scripts can be created for separator zones, form identification zones, or index zones.

 You can create scripts only for custom recognition profiles. You cannot create scripts for full text OCR or page-level form identification profiles.

1. On the **Tools** tab, in the **Recognition** group, click **Scripts**.
The **Recognition Script** window appears.
2. On the **Recognition profiles** list, select the custom recognition profile for which you want to create a script.
3. In the **Scripting language** box, select the scripting language. In this case, select **Visual C#**.
4. Click **Create**.

A new script is created and displayed in the Visual C# code editor. The file name assigned to the script appears in the title bar.

5. When finished writing your code, compile the script, exit the editor, and close the **Recognition Script** window.

You must compile the script and publish your batch class before the script can be used in batches. Your script cannot be applied to a batch created before the new publication date. In addition, you must republish if you make changes to the script.

Debugging Your Settings with a Visual C# Recognition Script

You can use a Visual C# recognition script to display the zone snippet when the batch is processed in the Recognition Server module. This is a good way to test your settings.

1. In the Administration module, create a recognition profile for a zone.
2. On the **Tools** tab, in the **Recognition** group, click **Scripts**.
The **Recognition Script** window appears.
3. Select a profile from the list of **Recognition** profiles.
4. Select a scripting language from the list of Scripting languages. In this case, select **Visual C#**.
5. Select **Create**.

A new script appears in the Visual C# code editor.

Sample Code

Publish the batch class and process a test batch through the Recognition Server module. For every zone to which the recognition profile is attached, a message displays the file name of the zone snippet, the value, and the confidence. If you have access to a viewer, you can display the zone snippet file to see the zone snippet image.

```
using Kofax.AscentCapture.NetScripting;
using Kofax.Capture.CaptureModule.InteropServices;
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;

namespace Tutorial__OMR
{
    public class Tutorial__OMR : RecognitionScript
    {
        public Tutorial__OMR()
            : base()
        {
            /*** Registers batch loading or unloading events
            this.BatchLoading += Tutorial__OMR_BatchLoading;
            this.BatchUnloading += Tutorial__OMR_BatchUnloading;
        }

        void Tutorial__OMR_RecognitionPostProcessing(object sender,
        PostRecognitionEventArgs e)
        {
            /*** Display a message box with the image file, value and confidence
            for each field
            MessageBox.Show(e.ImageFile + " " + e.Value + " " +
            e.Confidence.ToString());
        }
    }
}
```

```

void Tutorial__OMR_BatchUnloading(object sender)
{
    /*** Un-Registers any extra recognition events and then batch
loading/unloading events
    this.RecognitionPostProcessing -=
Tutorial__OMR_RecognitionPostProcessing;
    this.BatchUnloading -= Tutorial__OMR_BatchUnloading;
    this.BatchLoading -= Tutorial__OMR_BatchLoading;
}

void Tutorial__OMR_BatchLoading(object sender, ref bool
ImageFileRequired)
{
    /*** Registers RecognitionPostProcessing event to display the
recognition result for each recognition object
    this.RecognitionPostProcessing +=
Tutorial__OMR_RecognitionPostProcessing;
    ImageFileRequired = true;
}
}
}

```

Sample Visual C# Recognition Script

The following recognition script demonstrates how to set recognition criteria to enhance a field that may display poorly. The confidence level is set to 75%. If the confidence level for the field falls below 75%, the document is sent to the Quality Control module.

```

void Tutorial__OMR_RecognitionPostProcessing(object sender, PostRecognitionEventArgs
e)
{
    /*** Throws an exception if confidence was less than 75 percent
    if (e.Confidence < 75)
    {
        throw new
Kofax.AscentCapture.NetScripting.FatalErrorException("Confidence value was less than
75 percent.")
    }
}
}

```

Removing a Recognition Script

You can remove a recognition script to restore the default recognition procedures.

Use the Administration module to remove recognition scripts. When you remove a recognition script, the source code file is not deleted from the Scripts folder. Although you cannot select the file to "reattach" it to a recognition profile, you can add the customizations to a new script. The easiest way to do this is to create a new recognition script and open the old recognition script in a text editor such as Notepad. Then, you can copy code from the old script to the new script.

1. On the **Tools** tab, in the **Recognition** group, click **Scripts**.
The **Recognition Script** window appears.
2. On the **Recognition** profiles list, select the recognition profile for which you want to restore the default processing.
3. Click **Remove**.
The script is no longer associated with the recognition profile.

Field script

A field script is a small program that can contain variables and functions needed for validating an index field with an associated field type.

A field script is invoked when there is no document validation script for a document class, and the field script can be referenced by other custom scripts for documents. A field script can have only one script language defined.

A field script can contain field formatting functions and event arguments associated with each event.

Field Script Events

Event	Description
FieldFormatting	Called for every field as every other field is exited. Allows the display of a field to differ from the stored value.
FieldPreProcessing	Called as the field is entered. Set <code>e.SkipMode = SaveAndSkipDocumentOrFolder</code> to save the field and move to the next document or folder. Set <code>e.SkipMode = SaveAndSkipField</code> to save the field and move to the next field.
FieldPostProcessing	Called as the field is exited. Set <code>e.SaveAndSkipDocument = true</code> to save the field and move to the next document.
FormatFieldEventArgs	This class represents the event arguments for the <code>FieldScript.FieldFormatting</code> event.
PreFieldEventArgs	This class represents the event arguments for the <code>FieldScript.FieldPreProcessing</code> event.
PostFieldEventArgs	This class represents the event arguments for the <code>FieldScript.FieldPostProcessing</code> event.

For details about each event, event arguments, and properties, refer to the *Tungsten Capture API Reference*.

Sample Visual C# Field Script

When you create a new field script, a new class is derived from the `FieldScript` class of the Tungsten Capture NET Scripting API. The name of the class is based on the field type name (all non-alphanumeric characters of the class name are replaced with underscore characters). The name of the class can be changed; however, the changed name is not automatically updated in the generated field script.

The following is a field script sample that changes all mixed case letters of a particular field to upper case letters for reading clarity.

```
using Kofax.AscentCapture.NetScripting;
using Kofax.Capture.CaptureModule.InteropServices;
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
```

```
namespace Alphanumeric_12
{
    public class Alphanumeric_12 : FieldScript {

        public Alphanumeric_12(bool bIsValidation, string strUserID, string strLocaleName)
        : base(bIsValidation, strUserID, strLocaleName)
        {
            /*** Register FieldFormatting event with the Capture Validation/
Verification modules
            this.FieldFormatting += Alphanumeric_12_FieldFormatting;
        }

        ~Alphanumeric_12()
        {
            /*** UnRegister FieldFormatting event with the Capture Validation/
Verification modules
            this.FieldFormatting -= Alphanumeric_12_FieldFormatting;
        }
        void Alphanumeric_12_FieldFormatting(object sender, FormatFieldEventArgs e)
        {
            /*** Converts the string to upper case
            string strUpper = this.IndexField.Value.ToUpper();
            /*** Sets the value to the upper case string.
            this.IndexField.Value = strUpper;
        }
    }
}
```

Chapter 4

Registration File Creation

Custom extensions (custom modules, setup OCX programs, and workflow agents) must be registered with Tungsten Capture before they can be used. The registration process is necessary so that Tungsten Capture recognizes the custom extension as valid.

Custom extension registration is a two-part process that requires you to do the following:

- Set up a registration file (.aex file) that defines the property settings for the custom extension. See [Format for the Registration File](#) for more information about the registration file.
- Register the custom extension using one of the following:
 - Administration module. For details, see [Using the Administration Module to Manage Extensions](#).
 - Tungsten Capture Extension Registration Utility, which is run from a command line. For details, see [Tungsten Capture Extension Registration Utility](#).

Prior to registration, the custom extension registration file (the executable or DLL file), and the optional setup OCX must be saved to `<Tungsten Capture installation folder>\Bin`.

Export connectors and runtime OCXs cannot be registered using the methods described in this chapter. For more information on export connectors, see [Creating an Export Connector](#). For more information on runtime OCXs, see [Custom Panels and Applications](#).

Format for the Registration File

The format for the registration (.aex) file is similar to that of a standard Windows .ini file. See [Sample Registration Files](#) for registration file samples.

Registration files offer great flexibility. You can have a single file that contains information for all your custom modules, workflow agents, or setup OCXs. Alternatively, you can have separate files for each custom extension. If you have a custom module or a workflow agent that requires a setup OCX, you must put the setup OCX information in the same registration file as the extension that uses it.

The registration file sections are described in the following sections.

[\[Modules\] Section](#)

To register one or more custom modules, the .aex file must contain a header section labeled `[Modules]` that lists the display name of each custom module defined in the file. Custom module display names cannot exceed 32 characters.

i You can register items listed in the *[Modules]* section with the Custom Module Manager available from the Administration module. Alternatively, you can register them with the Tungsten Capture Extension Registration Utility (RegAscEx.exe), which is available from the Tungsten Capture `Bin` folder.

[Module Name] Section

For each custom module listed in the *[Modules]* section, the .aex file must include a corresponding *[Module Name]* section (labeled with the custom module name) that includes the property settings required to register the module. Each *[Module Name]* section can include the keys listed in the table. Note that some of the keys are required. If a required key is missing, an error occurs during registration.

Key	Description	Required
Description	Description of the purpose of the custom module. The description displays in the Queues tab on the Create Batch Class and Batch Class Properties windows for the selected module. Maximum length is 250 characters.	Yes
Follow	Indicates the processing function that the custom module function must follow in the batch class workflow. The following values are valid for the Follow key: Scan (default value) Document Separation Form Identification Automatic Index Validation Verification Custom modules are not allowed to follow the Export module. To specify proper queue ordering, you can set values for Follow and/or Precede or set a value for Function. If you specify a value for Follow or Precede, do not set a value for Function. If you do, errors occur when you attempt to register a custom module.	

Key	Description	Required
Function	<p>Specifies the function to be performed by the custom module. The defined function is used to perform publish checks and to ensure proper queue ordering. One or more of the following functions may be specified, delimited by semicolons.</p> <ul style="list-style-type: none"> Document Separation Page Registration Form Identification Automatic Index Validation Verification OCR Full Text Quality Control Other (default value) <p>To specify proper queue ordering, you can set a value for Function or set values for the Follow and/or Precede keys. If you specify a value for Function, do not set Follow or Precede. If you do, errors occur when you attempt to register the custom module.</p>	No
IconFile	<p>Path to a file containing the icon associated with the module name. A semicolon and the numeric index of the desired icon in the file may optionally follow the path. If this property is not specified, the default icon for the RuntimeProgram file is used. If no icon is found in either file, the module name appears without an icon. Maximum length for the icon file and path is 250 characters.</p>	No
ModuleID	<p>Unique identifier for a custom module. Used to synchronize information about custom modules when data is exported and imported between databases. Maximum length is 250 characters.</p> <p>This identifier must be unique across both custom modules and workflow agents.</p>	Yes

Key	Description	Required
Precede	<p>Indicates the processing function that the custom module function must precede in the batch class workflow. These Precede key values are valid:</p> <ul style="list-style-type: none"> Document Separation Form Identification Automatic Index Validation Verification OCR Full Text Export (default value) <p>Custom modules are not allowed to precede the Scan module.</p> <p>To specify proper queue ordering, you can set values for Follow and/or Precede OR set a value for Function. If you specify a value for Follow or Precede, do not set a value for Function. If you do, errors occur when you attempt to register a custom module.</p>	No
Runtime Command Line	Specifies command line parameters that must be passed to the custom module's runtime program when it is invoked from Batch Manager. Maximum length is 250 characters.	No
Runtime Program	<p>The runtime executable for the custom module. This executable is invoked when a batch in the ready state is selected in Batch Manager. Maximum length is 250 characters.</p> <p>For each workstation that will use it, the executable must exist in <code><Tungsten Capture installation folder>\Bin</code>.</p>	Yes
SetupProgram	Specifies a <code>[Setup]</code> Section within the .aex file that defines the properties of the setup module for a custom module. The name of this section is also the display name of the custom module. Maximum length is 250 characters. See the [Setup] Section for more information.	No
SupportsNonImageFiles	Specifies whether the custom module supports non-image files (eDocuments). Setting this key to True turns on non-image file support for the custom module. Setting this key to False or omitting the key turns off non-image file support for the custom module.	No
SupportsTableFields	Specifies whether the customization supports tables. Setting this key to True turns on table support for the customization. Setting this key to False turns off table support for the customization. If not specified, the default setting is False.	No
SuppressBatchContents	<p>Specifies whether the custom standard module includes the Batch Contents panel. Setting this key to True removes the panel. Setting this key to False, or omitting the key, causes the Batch Contents panel to be available.</p> <p>This key applies only to custom standard modules.</p>	No

Key	Description	Required
SuppressBatchFilters	Specifies whether the custom standard module includes the Batch Filters toolbar. Setting this key to True removes the toolbar. Setting this key to False, or omitting the key, causes the Batch Filters toolbar to be available. This key applies only to custom standard modules.	No
SuppressBatchNavigation	Specifies whether the custom standard module includes the Batch Navigation toolbar. Setting this key to True removes the toolbar. Setting this key to False, or omitting the key, causes the Batch Navigation toolbar to be available. This key applies only to custom standard modules.	No
SuppressBatchThumbnails	Specifies whether the custom standard module includes the Batch Thumbnails panel. Setting this key to True removes the panel. Setting this key to False, or omitting the key, causes the Batch Thumbnails panel to be available. This key applies only to custom standard modules.	No
SuppressBatchTools	Specifies whether the custom standard module includes the Batch Tools toolbar. Setting this key to True removes the toolbar. Setting this key to False, or omitting the key, causes the Batch Tools toolbar to be available. This key applies only to custom standard modules.	No
Suppress ImageTools	Specifies whether the custom standard module includes the Image Tools toolbar. Setting this key to True removes the panel. Setting this key to False, or omitting the key, causes the Image Tools toolbar to be available. This key applies only to custom standard modules.	No
Suppress ImageViewer	Specifies whether the custom standard module displays the Tungsten Capture Image Viewer. Setting this key to True hides the image viewer. Setting this key to False, or omitting the key, causes the image viewer to be visible. This key applies only to custom standard modules.	No
Suppress Notes	Specifies whether the custom standard module includes the Notes panel. Setting this key to True removes the panel. Setting this key to False, or omitting the key, causes the Notes panel to be available. This key applies only to custom standard modules.	No
SuppressScanControls	Specifies whether the custom standard module includes the Scan Controls panel. Setting this key to True removes the panel. Setting this key to False, or omitting the key, causes the Scan Controls panel to be available. This key applies only to custom standard modules.	No

Key	Description	Required
UsesExtendedRecognition Info	<p>To enable communications between custom modules, the Recognition Server is able to generate the extended recognition data in the form of an XML string. By default, the Recognition Server does not output this XML data unless a custom module is registered to indicate otherwise. Extended recognition data can be very large and have a significant impact on performance and/or disk space.</p> <p>If a batch class contains any custom module with this flag set to true, extended recognition information is generated in the Recognition Server.</p> <p>Example: [Sample] RuntimeProgram=CMSample.EXE ModuleID=Kofax.Sample Description=This Sample module supports non-image files Version=1.0 UsesExtendedRecognitionInfo=True</p> <p>The default, if this value is not provided, is <i>False</i>.</p> <p>A document-based custom storage string, named <i>Kofax.AC.ExtendedRecognitionInfo</i> is used for the storage of extended recognition data.</p> <p>The Recognition Server generates values for the this custom storage string whenever it produces an index field value result generated by the Tungsten Automation High Performance OCR Zonal, Tungsten Automation High Performance ICR Zonal, or the Tungsten Automation Advanced OCR Zonal engines, and a custom module has registered itself as using extended recognition information.</p> <p>Index Fields with values assigned by other means (such as OMR, or Tungsten Automation OCR) do not cause the generation of ExtIndexField elements within the string.</p> <p>The Recognition Server updates only those ExtIndexField elements where the corresponding index field value is updated.</p> <p>The "Kofax.AC.ExtendedRecognitionInfo" custom storage string is never cleared, even when the index value is changed.</p>	No
Version	Custom module version number assigned by the developer. If this property is not specified, no version number appears.	No

[Workflow Agents] Section

To register one or more workflow agents, the .aex file must contain a header section labeled [Workflow Agents] that lists the display name of each workflow agent. Workflow agent display names cannot exceed 32 characters.

You can register the items listed in the *[Workflow Agents]* section with the Workflow Agent Manager available from the Administration module, or with the Tungsten Capture Extension Registration Utility (RegAscEx.exe), which is available from the Tungsten Capture `Bin` folder.

[Workflow Agent Name] Section

For each workflow agent listed in the *[Workflow Agents]* section, the .aex file must include a corresponding *[Workflow Agent Name]* section (labeled with the workflow agent name) that includes the property settings required to register the agent. Each *[Workflow Agent Name]* section can include the keys listed below. Note that some of the keys are required. If a required key is missing, an error occurs during registration.

Key	Description	Required
Description	Description of the purpose of the workflow agent. Maximum length is 250 characters.	Yes
SetupProgram	Specifies a [Setup] Section within the .aex file that defines the properties of the setup program for a workflow agent. Maximum length is 250 characters.	No
Version	Workflow agent version number assigned by the developer. If this property is not specified, no version number appears.	No
WorkflowAgentFile	.Dll that contains the Workflow Agent runtime COM server.	Yes
WorkflowAgentID	Value that indicates a unique identifier for the Workflow Agent. Maximum length is 250 characters. This identifier must be unique across both Workflow Agents and Custom Modules.	Yes
WorkflowAgentProgID	The runtime COM ProgID of the Workflow Agent.	Yes
WorkflowAgentSkipIfCantLoad	By default, the system places the batch in error if it cannot load the runtime COM server on a particular station. However, with this flag set, the system does not place the batch in error if the Workflow Agent does not exist, and instead continues with the default batch workflow. To set this flag, include WorkflowAgentSkipIfCantLoad=true in the .aex file.	No

[Setup Programs] Section

To register one or more setup OCXs, the .aex file must contain a *[Setup Programs]* section that lists the names of all *[Setup]* sections defined in the file. See [\[Setup\] Section](#) for more information.

If a setup program is listed in *[Setup Programs]*, but not defined in its own *[Setup]* section, an error occurs during registration.

You must register Items listed in the *[Setup Programs]* section with the Tungsten Capture Extension Registration Utility (RegAscEx.exe) available from the Tungsten Capture `Bin` folder. You cannot register them from the Administration module.

[Setup] Section

For each item listed in the [Setup Programs], [Module Name], or [Workflow Agent Name] sections, the .aex file must include a corresponding [Setup] section (labeled with the item name) that defines the setup program.

Each [Setup] section can include the keys listed below. Note that some of the keys are required. If a required key is missing, an error occurs during registration.

Key	Description	Required
BatchClassMenus	Names one or more [Menu] sections in the .aex file that define the menu items to be added to the context menu for batch class nodes. Multiple items must be delimited by semicolons. The name of the section is also the internal name of the menu (passed to the OCX on ActionEvents). Maximum length is 250 characters.	No
DocumentClassMenus	Same as BatchClassMenus, except that it defines menu items to add to the context menu for document class nodes.	No
FieldTypeMenus		No
FormIdZoneMenus		No
FormTypeMenus		No
IndexGroupMemberZoneMenus		No
IndexGroupMemberZonesMenus		No
IndexZoneMenus		No
IndexZonesMenus		No
InitSizeX	The initial horizontal size of the panel in screen resolution pixels. The default is 50.	No
InitSizeY	The initial vertical size of the panel in screen resolution pixels. The default is 50.	No
ManagedApi	A value of 1 indicates that the custom module uses the Managed .NET APIs for Tungsten Capture 10.1 or later; a value of 0 indicates that the custom module uses VB6 Interop APIs.	No
MenuBarMenu	Names one [Menu Bar] section that defines the tab to add to the Ribbon. Multiple items are not allowed. Maximum length is 250 characters.	No
MinSizeX	The minimum horizontal size of the panel when undocked in screen resolution pixels. The default is 50.	No
MinSizeY	The minimum vertical size of the panel when undocked in screen resolution pixels. The default is 50.	No

Key	Description	Required
OCXFile	Path, including the file name, of the Setup OCX file. Maximum length is 250 characters. If no path exists, <Tungsten Capture installation folder>\Bin is used.	Yes
PageLevelBarcodeMenus	Same as BatchClassMenus, except that it defines the menu items to add to the context menu for page level bar code nodes.	No
PageLevelBarcodesMenus	Same as BatchClassMenus, except that it defines the menu items to add to the context menu for page level bar code collection nodes.	No
ProgID	The COM program ID of the module specified with the OCXFile key. Maximum length is 250 characters.	No
RegistrationZoneMenus	Same as BatchClassMenus, except that it defines the menu items to add to the context menu for registration zone nodes.	No
RegistrationZonesMenus	Same as BatchClassMenus, except that it defines the menu items to add to the context menu for registration zone collection nodes.	No
SamplePageMenus	Same as BatchClassMenus, except that it defines the menu items to add to the context menu for sample page nodes.	No
SeparationZoneMenus	Same as BatchClassMenus, except that it defines the menu items to add to the context menu for separation zone nodes.	No
Visible	A value of 1 indicates visible; 0 indicates invisible. If set to 0, the panel is not initially displayed and the DisplayName does not appear. The default is 1. This key is ignored when registering a custom module from the command line.	No

[Menu] Section

A [Menu] section defines the text for a command listed in the [Menu Bar] and [Setup] sections. A [Menu] section has one required key. If the key is missing, an error occurs during registration.

[Menu] Section Key

Key	Description	Required
MenuText	Display text for the command.	Yes

[Menu Bar] Section

For each tab listed in a [Setup] section, the .aex file must include a corresponding [Menu Bar] section (labeled with the tab name) that defines the tab. Each [Menu Bar] section must contain the keys listed below. Note that all keys are required. If a key is missing, an error occurs during registration.

[Menu Bar] Section Keys

Key	Description	Required
MenuBarText	Displays text for the tab.	Yes
Menus	Names one or more [Menu] sections that define the commands to be added to this tab. Multiple commands are delimited by semicolons. The name of the section is also the internal name of the tab (passed to the OCX on ActionEvents). Maximum length is 250 characters.	Yes

Sample Registration Files

This section contains several samples that show the file structure for registration files.

Sample 1: Registering Two Custom Modules

The sample registration file shown here specifies two custom modules: ABC Image Cleanup and XYZ Index. ABC Image Cleanup does not have a custom setup module, but XYZ Index does. The XYZ Index batch class context menu has two custom menu items: XYZ Properties and XYZ Reset to Default. The Ribbon has a custom tab item, XYZ Index, with two commands: XYZ Item 1 and XYZ Item 2. By default, the group is called "Menu Items."

Comments, which are shown in bold text, are not part of the registration file.

```
[Modules] - Modules Section
ABC Image Cleanup
XYZ Index

[ABC Image Cleanup] - Module Name Section
ModuleID=ABC.ImageCleanup
Version=1.0
RuntimeProgram=ABCImage.exe
Description=The ABC Image Cleanup module performs image enhancement

[XYZ Index] - Module Name Section
ModuleID=XYZ.Index
IconFile=XYZ.ico
RuntimeProgram=XYZIndex.exe
RuntimeCommandline=/k
SetupProgram=XYZ Setup
Description=The XYZ Index module automatically gathers index data
Function=Automatic Index
```

```
[XYZ Setup] - Setup Section
OCXFile=XYZSetup.ocx
ProgID=XYZCorp.IndexSetup
InitSizeX=200
InitSizeY=100
Visible=1
BatchClassMenus=XYZ Batch Menu 1;XYZ Batch Menu 2
MenuBarMenu=XYZ Menu Bar

[XYZ Batch Menu 1] - Menu Section
MenuText=XYZ Properties

[XYZ Batch Menu 2] - Menu Section
MenuText=XYZ Reset to Default

[XYZ Menu Bar] - Menu Bar Section
MenuText=XYZ Index
Menus=XYZ Bar Menu1;XYZ Bar Menu 2

[XYZ Bar Menu 1] - Menu Section
MenuText=XYZ Item 1

[XYZ Bar Menu 2] - Menu Section
MenuText=XYZ Item 2
```

Sample 2: Defining a Tab on the Ribbon

The following custom module registration file (.aex file) defines one tab on the Ribbon, including the name of the tab and two commands.

Comments, which are shown in bold text, are not part of the registration file.

 The sample requires SmpSetUp.ocx to be located in the root folder of drive C.

```
[Setup Programs] - Setup Programs Section
Sample Setup

[Sample Setup] - Setup Section
OCXFile=c:\SmpSetUp.ocx
ProgID=SampleSetUp.SmpSetUp
Visible=1
MinSizeX=50
MinSizeY=50
InitSizeX=432
InitSizeY=351
MenuBarMenu=XYZ Menu Bar

[XYZ Menu Bar] - Menu Bar Section
MenuBarText=XYZ Index
Menus=XYZ Bar Menu 1;XYZ Bar Menu 2

[XYZ Bar Menu 1] - Menu Section
MenuText=XYZ menu item 1

[XYZ Bar Menu 2] - Menu Section
MenuText=XYZ menu item 2
```

Sample 3: Defining Context Menu Items

The following custom module registration file defines custom context menu items for nodes in the tree view available from the Administration module.

Comments, which are shown in bold text, are not part of the registration file.

```
[Setup Programs] - Setup Programs Section
Sample Setup

[Sample Setup] - Setup Section
OCXFile=c:\SmpSetUp.ocx
ProgID=SampleSetUp.SmpSetUp
Visible=1
MinSizeX=50
MinSizeY=50
InitSizeX=432
InitSizeY=351
MenuBarMenu=XYZ Menu Bar
BatchClassMenus=BatchClass 1
DocumentClassMenus=DocClass 1
FormTypeMenus=FormType 1
SamplePageMenus=SamplePage 1
SeparationZoneMenus=Separation 1
RegistrationZoneMenus=RegZone 1
IndexZoneMenus=IndexZone 1
FormIdZoneMenus=FormId 1
FieldTypeMenus=FieldType 1
PageLevelBarcodeMenus=PLB 1
IndexZonesMenus=Index Zones 1
PageLevelBarcodesMenus=PLBs 1
RegistrationZonesMenus=Regzones 1
IndexGroupMemberZoneMenus=GroupMember 1
IndexGroupMemberZonesMenus=GroupParent 1

[XYZ Menu Bar] - Menu Bar Section
MenuBarText=XYZ Index
Menus=XYZ Bar Menu 1;XYZ Bar Menu 2

[XYZ Bar Menu 1] - Menu Section
MenuText=XYZ menu item 1

[XYZ Bar Menu 2] - Menu Section
MenuText=XYZ menu item 2

[BatchClass 1] - Menu Section
MenuText=I show up on the &BatchClass tree node

[DocClass 1] - Menu Section
MenuText=I show up on the &DocClass tree node

[FormType 1] - Menu Section
MenuText=I show up on the &FormType tree node

[SamplePage 1] - Menu Section
MenuText=I show up on the &SamplePage tree node

[Separation 1] - Menu Section
MenuText=I show up on the Separation &Zones tree node

[RegZone 1] - Menu Section
```

```

MenuText=I show up on the &Reg Zone tree node

[IndexZone 1] - Menu Section
MenuText=I show up on the &Index Zone tree node

[FormId 1] - Menu Section
MenuText=I show up on the Form&Id tree node

[FieldType 1] - Menu Section
MenuText=I show up on the &FieldType tree node

[PLB 1] - Menu Section
MenuText=I show up on the Page level bar code tree node

[Index Zones 1] - Menu Section
MenuText=I show up on the &Index Zones tree node

[PLBs 1] - Menu Section
MenuText=I show up on the Page level bar code parent tree node

[Regzones 1] - Menu Section
MenuText=I show up on the Regzones tree node

[GroupParent 1] - Menu Section
MenuText=I show up on the Group&Parent tree node

[GroupMember 1] - Menu Section
MenuText=I show up on the Group&Member tree node

```

Sample 4: Defining a Workflow Agent

The following workflow agent registration file defines a workflow agent with a custom context menu item available from the Administration module.

Comments, which are shown in bold text, are not part of the registration file.

```

[Workflow Agents] - Workflow Agents Section
Validation Workflow Agent

[Validation Workflow Agent] - Workflow Agent Name Section
WorkflowAgentID=Kofax.AgentWithOCX
WorkflowAgentProgID=WFAgent.SampleWorkflowAgent
WorkflowAgentFile=WFAgent.dll
Description=This sample workflow agent is combined with a setup OCX.
Version=7.5
SupportsNonImageFiles=True
SetupProgram=Workflow Agent Setup

[Workflow Agent Setup] - Setup Section
OCXFile=SampleWorkflowOcx.ocx
ProgID=SampleWorkflowOcx.SampleWorkflow
Visible=0
MinSizeX=300
MinSizeY=150
BatchClassMenus=Workflow Agent Test Menu

[Workflow Agent Test Menu] - Menu Section
MenuText=Validation &Workflow Properties...

```

Using the Administration Module to Manage Extensions

You can use the Administration module to register custom modules and workflow agents. You can also use the command line registration utility, which is explained in [Tungsten Capture Extension Registration Utility](#).

You cannot use the Administration module to register setup OCXs, you must use the Tungsten Capture Extension Registration Utility.

Custom Module Management

The following procedures cover registering and removing custom modules.

Registering a Custom Module

Before registration, make sure to place the following files in <Tungsten Capture installation folder>\Bin:

- Setup OCX for the custom extension (optional)
 - Executable for the custom extension
 - Registration (.aex) file for the custom extension
1. Start the Administration module.
 2. On the **Tools** tab, in the **System** group, click **Custom Modules**.
The **Custom Module Manager** window appears.
 3. On the **Custom Module Manager** window, click **Add**.
 4. On the Open window, browse to <Tungsten Capture installation folder>\Bin, and select the .aex file associated with the custom module to register.
 5. Click **Open**.
The custom modules listed in the [Modules] section of the .aex file appear in the **Custom Modules** window.
 6. Select the name of the modules to register and click **Install**.
A confirmation message appears to inform you when the registration is successful.
 7. Click **OK** to clear the message, and then click **Close** to exit the **Custom Modules** window.
The name of each newly registered module appears in the **Custom Module Manager** window. Also, each newly registered custom module is added to the list of **Available Queues** in the **Queues** tab on the **Create Batch Class** and **Batch Class Properties** windows.

Viewing Properties for a Custom Module

The Custom Module Properties and Workflow Agent Properties windows list all the registered settings for the selected application. The settings are based on the information defined in the .aex file for the custom extension. The properties are listed for reference purposes only; you cannot edit them.

1. On the **Custom Module Manager** window, select a custom module and click **Properties**.

The **Custom Module Properties** window appears. On the **General** tab, you can view information that describes the custom module.

2. Click the **Programs** tab.

On this tab, you can view the location of the runtime file for the module, along with the location of the optional setup program.

3. Click the **Advanced** tab.

On this tab, you can view the functions defined for the custom module.

The **Advanced** tab also lists the valid processing order allowed for the custom module function. The "Follow" entry lists the processing function that the custom module function must follow. The "Precede" entry lists the function that the custom module must precede. The selections for Follow/Precede affect the valid order that is allowed for the custom module on the **Queues** tab on the **Create Batch Class** and **Batch Class Properties** windows.

Removing a Custom Module

You can use the Administration module to remove or "unregister" a custom module from Tungsten Capture. Before removing the custom module, you must ensure that it is not used in any published batch class. Otherwise, Tungsten Capture prevents you from removing it.

1. Start the Administration module.

2. On the **Tools** tab, in the **System** group, click **Custom Modules**.

The **Custom Module Manager** window appears.

3. Select the name of the custom module to remove.

4. Click **Remove**.

If the selected custom module is not used in any published batch class, it is cleared from the list. The **Custom Module Manager** does not allow you to remove a custom module that is used by any batch classes.

5. Click **Close**.

Workflow Agent Management

The following procedures explain how to register and remove workflow agents.

Registering a Workflow Agent

Before registration, make sure that you have placed the following files in <Tungsten Capture installation folder>\Bin:

- Setup OCX for the custom extension (optional)
- Executable for the custom extension
- Registration (.aex) file for the custom extension

1. Start the Administration module.

2. On the **Tools** tab, in the **System** group, click **Workflow Agents**.

The **Workflow Agent Manager** window appears.

3. On the **Workflow Agent Manager** window, click **Add**.

4. On the Open window, browse to <Tungsten Capture installation folder>\Bin, and select the .aex file associated with the workflow agent to register.

5. Click **Open**.

Each workflow agent listed in the [Workflow Agents] section of the .aex file appears in the **Workflow Agents** window.

6. Select the name of each workflow agent to register and click **Install**.

A confirmation message appears to inform you when the registration is successful.

7. Click **OK** to clear the message, and then click **Close** to exit the **Workflow Agents** window.

The name of each newly registered agent appears in the **Workflow Agents Manager** window. Also, each newly registered agent is added to the list of workflow agents in the **Queues** tab on the **Create Batch Class** and **Batch Class Properties** windows.

Viewing Properties for a Workflow Agent

The Workflow Agent Properties windows list all of the registered settings for the selected workflow agent. The settings are based on the information defined in the .aex file. The properties are listed for reference purposes only; you cannot edit them.

1. On the **Workflow Agent Manager** window, select a Workflow Agent and click **Properties**.

The **Workflow Agent Properties** window appears. On the **General** tab, you can view information that describes the Workflow Agent.

2. Click the **Programs** tab.

On this tab, you can view the location of the runtime file for the module, along with the location of the optional setup program.

Removing a Workflow Agent

Use the Administration module to remove or "unregister" a workflow agent from Tungsten Capture. Before removing the workflow agent, you must ensure that it is not used in any published batch class. Otherwise, Tungsten Capture prevents you from removing it.

1. Start the Administration module.

2. On the **Tools** tab, in the **System** group, click **Workflow Agents**.

The **Workflow Agents** window appears.

3. Select the name of the workflow agent to remove.

4. Click **Remove**.

If the selected workflow agent is not used in any published batch class, it is cleared from the list. The **Workflow Agent Manager** does not allow you to remove a workflow agent that is in use by any batch classes.

5. Click **Close**.

Tungsten Capture Extension Registration Utility

The Tungsten Capture Extension Registration Utility (RegAscEx.exe) is a standalone console application that registers and unregisters custom modules, workflow agents, and setup OCXs. For custom modules and workflow agents, this utility is an alternative to the windows that are available from the Administration module. With the registration utility, you also can incorporate the registration process into the custom extension installation program. RegAscEx.exe resides in <Tungsten Capture installation folder>\Bin.

The utility can be used to register:

- Custom modules
- Custom modules with setup OCXs
- Setup OCXs
- Workflow agents
- Workflow agents with setup OCXs

The registration utility, which has no graphical user interface, is controlled completely via command line parameters.

Prior to registration, you must save the .aex file, executable file, and the optional setup OCX to *<Tungsten Capture installation folder>\Bin*.

i The Visible key available for [Setup] sections is ignored if you register a custom extension with this utility prior to running the Administration module. This happens because the Administration module hides the custom panel if it is launched after you register the custom panel. If you launch the Administration module prior to using the command line to register the custom module, the custom module should appear as expected.

Command Line Parameters

The utility supports several command line parameters to specify which custom extensions to register or unregister, an output file, and silent mode operation. If any invalid or incomplete parameters are specified, you are prompted with the correct usage, as shown in the figure.

Input File [/f {file name}]

You must specify a file that contains the registration settings for one or more custom extensions to register or unregister in the Tungsten Capture database. The /f flag is followed by the .aex file name, as in the following examples:

```
RegAscEx /f custom.aex (for a custom module and/or workflow agent)
RegAscEx /f SetupOCX.aex (for a setup program)
```

Every custom extension defined in the specified file is processed. Status messages are displayed to the console for each application registered or unregistered. See [Format for the Registration File](#) for more information.

Output File [/o {file name}]

You may optionally specify an output file where all messages and statuses are recorded. The messages are output to the file and to the default display of the console. The /o flag is followed by the output file name, as in the following example:

```
RegAscEx /f custom.aex /o output.log
```

Module Name [/m {module name}]

You may optionally specify a single custom module from the input file to register or unregister. Only the specified custom module is processed, rather than the default behavior of processing all custom

modules in the input file. The `/m` flag is followed by the custom module name enclosed in quotation marks, as in the following example:

```
RegAscEx /f custom.aex /m "XYZ Image Processing"
```

Workflow Agent Name [`/w {workflow agent name}`]

You may optionally specify a single custom module from the input file to register or unregister. Only the specified custom module is processed, rather than the default behavior of processing all custom modules in the input file. The `/m` flag is followed by the custom module name enclosed in quotation marks, as in the following example:

```
RegAscEx /f custom.aex /m "XYZ Image Processing"
```

Setup Programs [`/x {setup program name}`]

You may optionally specify a single setup program from the input file to register or unregister. Only the specified setup program is processed, rather than the default behavior of processing all Tungsten Capture extensions in the input file. The `/x` flag is followed by the setup program name enclosed in quotation marks, as in the following example:

```
RegAscEx /f SetupOCX.aex /x "Setup OCX"
```

Runtime Programs [`/r {runtime program name}`]

You may optionally specify a single runtime program from the input file to register or unregister. Only the specified runtime program is processed, rather than the default behavior of processing all Tungsten Capture extensions in the input file. The `/r` flag is followed by the runtime program name enclosed in quotation marks, as in the following example:

```
RegAscEx /f RuntimeOCX.aex /r "Runtime OCX"
```

Unregister [`/u`]

You may optionally use the `/u` flag to unregister one or more custom modules in the Tungsten Capture database, as in the following example:

```
RegAscEx /f custom.aex /u
```

If the `/m` parameter is also used, only the specified custom module is unregistered. If the `/w` parameter is also used, only the specified workflow agent is unregistered. If the `/x` parameter is also used, only the specified setup program is unregistered. Otherwise, all custom extensions specified in the `.aex` file are unregistered. If any batch class is using a custom extension, the application cannot be unregistered. A warning message is displayed, as follows:

```
<Application Name> is being used by the following batch
classes and cannot be removed.
```

This message is followed by a list of up to 10 batch classes that use the custom extension. If more than 10 batch classes are detected, the first nine are listed, followed by the phrase "and more."

Silent Mode [/s]

You may request that the registration utility operate without generating any output messages to the console by adding the /s command line parameter, as in the following example:

```
RegAscEx /f custom.aex /s
```

If an output file is specified using the /o parameter, that file is still generated with all output messages.

Usage [/?]

The " /?" parameter displays the proper usage of the registration utility to the console. It defines each of the supported command line parameters and provides an example for formatting the command line. For example, you can use the following to display information about proper usage:

```
RegAscEx /?
```

Input

The exact format of the .aex file is documented in [Format for the Registration File](#). The file contains a header section that lists the name of each custom extension to be defined. Subsequent sections itemize the properties for each custom extension to be registered. The Tungsten Capture Extension Registration Utility verifies the values for the properties that are required in the .aex file.

Output

This section contains messages that may be displayed by the Tungsten Capture Extension Registration Utility. The following examples assume that silent mode operation (using the /s command line parameter) has not been specified. Console messages also appear in the output file if they are specified using the /o command line parameter.

Proper Usage

If you call the Tungsten Capture Extension Registration Utility with the " /?" parameter or specify an invalid command line parameter, the valid parameter information appears.

Error and Warning Messages

If the utility fails to register or unregister a custom extension, an error or warning message appears to explain the problem. If an invalid parameter is specified or a required parameter is missing, the proper usage prompt also appears.

Chapter 5

Workflow Agent Creation

A workflow agent is a custom application you can create to examine and modify batch data, change the routing of a batch, restrict access to a batch in certain modules, and obtain the status of a batch. Workflow agents are attached at the batch class level and invoked whenever a batch is closed from a module.

This chapter describes how you can create, install, and register a workflow agent to customize your Tungsten Capture process. A sample workflow agent written in Visual Basic .NET is provided later in this chapter.

In this chapter, you will learn to:

- Design a workflow agent
- Implement an optional setup OCX for the workflow agent
- Write the runtime module using Visual Basic .NET
- Create the workflow agent registration file
- Install and register the workflow agent
- Remove the workflow agent

Workflow Agent Design

The purpose of a custom workflow agent for an assigned batch class is to modify the typical processing of a batch in Tungsten Capture. When designing the workflow agent, consider the following questions:

- What is the purpose of the workflow agent?
- What functions should be performed by the workflow agent to carry out its purpose?
- Will an administrator need to have configuration options? If so, a setup OCX is needed.

Setup OCX Implementation

A setup OCX for the workflow agent is optional, but if your workflow agent requires configuration prior to running, a setup OCX should be considered. When creating the setup OCX, consider the following:

- What the setup OCX interface will look like
- Where the configuration settings will be stored
- How the workflow agent is configured

The setup OCX, which provides the configuration properties for the workflow agent through the Administration module, can be displayed via toolbars, panels, or context menus. These properties are available only for batch classes that are assigned the workflow agent.

See [Setup OCX Creation](#) for more information about coding a custom setup OCX.

Writing the Runtime Module

Now that you've determined the purpose of your custom workflow agent and defined its functions and goals, you are ready to code your application. The sample workflow agent provided later in this chapter is written in Visual Basic .NET.

You can write a workflow agent using any of the supported programming languages and development environments:

- Microsoft Visual Studio version 2022 (Community, Professional, and Enterprise editions)
- Microsoft Visual Studio version 2019 (Community, Professional, and Enterprise editions)
- Microsoft Visual Studio version 2017 (Community, Professional, and Enterprise editions)
- Microsoft Visual Studio version 2015 (Community, Professional, and Enterprise editions)
- Microsoft Visual Studio version 2013 (Express, Professional, and Enterprise editions)
- Microsoft Visual Studio version 2012 (Express, Professional, and Enterprise editions)

You also need the .NET Framework 4.8 Developer Pack installed in your development environment. If it is not already installed, the .NET Framework 4.8 Developer Pack can be installed via Visual Studio 2022 or separately by downloading it from Microsoft site.

Also, you must use one of the development environments above to compile against Tungsten Capture 11.1 libraries, which are targeted to .NET 4.8. See [Backward Compatibility](#) for more information.

Code Project Settings

When generating your Visual Basic project, the following must be set:

- Workflow agent must be a registered COM server.
- Code project must reference the Tungsten Capture Custom Workflow .NET Type Interface Library and the Tungsten Capture Custom Workflow .NET Type Implementation Library
- Workflow agent COM class must implement the IACWorkFlowAgent interface.

The workflow agent application named DeletePageWFA is used as a sample later in this chapter, and sections of the code that perform certain functions are described. The sample application:

- Ensures that only even-numbered pages are deleted. If odd-numbered pages are marked for deletion, the batch is sent to Quality Control.
- References a setup OCX, which adds a new menu item to the Batch Class context menu. This menu item gives the administrator the ability to specify whether the workflow agent will check for even-numbered page deletion. The sample code for the setup OCX is provided and described in [Setup OCX](#).

Tungsten Capture API Library References

The sample custom workflow agent uses the objects, methods, and properties defined in the following API libraries:

- Tungsten Capture Custom Workflow .NET Type Interface Library (Kofax.Capture.SDK.Workflow.dll)
- Tungsten Capture Custom Workflow .NET Type Implementation Library (Kofax.Capture.ACWFLib.dll)
- Tungsten Capture Optimized Custom Module .NET Type Interface Library (Kofax.Capture.SDK.Data.dll)
- Tungsten Capture Optimized Custom Module .NET Type Implementation Library (Kofax.DBLiteOpt.dll)

Set your Visual Basic project to reference these libraries.

For more information about the Tungsten Capture Custom Workflow .NET Type Interface Library (and the Tungsten Capture Optimized Custom Module .NET Type Interface Library), see the *Tungsten Capture API Reference* in the `Documentation\Help\APIRef` folder, which is available from your Tungsten Capture installation media.

Project Property Settings

Set your Visual Basic project properties to the following:

- Unattended Execution
- Retained in Memory

Workflow Agent Sample

The sample workflow agent is part of a more comprehensive sample that includes a custom workflow agent, custom setup OCX for the workflow agent, custom panel, and custom module.

Comprehensive sample:

1. Uses a workflow agent setup OCX that gives the administrator the ability to decide whether to remove even-numbered pages for a specified batch class.
2. Has a batch for the batch class created and scanned.
3. Provides a custom panel that the operator uses to mark even-numbered pages for deletion.
4. Has the workflow agent check the pages of the batch for odd-numbered pages that were incorrectly marked for deletion. These incorrectly marked pages are reset, and a page note is added indicating that the page should not have been marked for deletion.
5. Runs the custom module, which deletes the even-numbered pages marked for deletion.
6. Displays the Quality Control module, which shows that the pages were reset.

The code for the sample workflow agent is provided in this section; however, the code for the setup OCX, custom panel, and the custom module are provided and described in [Setup OCX Creation](#), [Custom Panels and Applications](#), and [Custom Module Creation](#), respectively.

Workflow Agent Program

The purpose of the sample custom workflow agent is to ensure that only even-numbered pages are marked for deletion. Odd-numbered pages that are marked for deletion are reset, sent to Quality Control, and flagged with a page note indicating that the page is not to be deleted.

The workflow agent application (DeletePageWFA) includes the major functions outlined below.

Inherit the IACWorkflowAgent Interface

Make sure that you have the line of code in your custom application that implements the `IACWorkflowAgent` interface. This line of code (shown in bold below) should be placed near the beginning of the `DeletePageWFA` application.

```
Imports System
Imports System.Runtime.InteropServices
Imports Kofax.Capture.SDK.Workflow
Imports Kofax.Capture.ACWFLib
Imports Kofax.Capture.DBLiteOpt
Imports Kofax.Capture.SDK.Data
Imports System.Windows.Forms

'*** You must generate a GUID and replace the GUID below
'*** Also, it is recommended that the PROGID be explicitly set
<GuidAttribute("6874CD00-1F18-4A7D-8F76-1F555E1F0F87"), _
ClassInterface(ClassInterfaceType.None), _
ProgId("VBNet.WFAgent"), _
CLSCompliant(False)>
Public Class VBNetWFAgent
    Implements Kofax.Capture.SDK.Workflow.IACWorkflowAgent
    Private m_oSetupDataElement As ACDataElement = Nothing
    Private strBatchClassName As String
    ...
End Class
```

ProcessWorkflow Function

The `ProcessWorkflow` function is the main method for the custom workflow agent. This subroutine is called whenever a batch is closed, rejected, or suspended. This implementation of the `ProcessWorkflow` function performs several functions:

- Checks if the batch is leaving the Scan module
- Extracts the runtime and setup data information
- Iterates through the pages and checks whether the page marked for deletion should indeed be deleted (that is, it is an even-numbered page)

Check if the Batch is Leaving the Scan Module

The `ProcessWorkflow` function checks if the batch is leaving the Scan module. If it isn't, the subroutine is exited. A check is also made to determine whether the batch is being suspended or if the batch is in error. If so, the subroutine is exited.

Extract the Runtime and Setup Data

A segment of the code extracts data from the following elements in preparation for reading each item:

- Setup ACDataElement
- Runtime ACDataElement
- Batch ACDataElement
- BatchClass ACDataElement
- Pages ACDataElement

Iterate Through the Documents

A segment of the code:

- Determines whether the pages should be checked (depending on whether the "Delete Even Page Setup" menu item is selected from the setup OCX context menu for the batch).
- Cycles through each page and increments the page count.

The ProcessWorkflow function code segment follows:

```
Public Sub ProcessWorkflow(ByRef oWorkflowData As _
Kofax.Capture.SDK.Workflow.IACWorkflowData Implements _
Kofax.Capture.SDK.Workflow.IACWorkflowAgent.ProcessWorkflow

'*** Let's make sure we are leaving Scan
If oWorkflowData.CurrentModule.Name = "Scan" Then

'*** Extract the Runtime DataElement
Dim oRTElem As ACDataElement
oRTElem = oWorkflowData.ExtractRuntimeACDataElement(0)

'*** Extract the Setup DataElement
m_oSetupDataElement = _
oWorkflowData.ExtractSetupACDataElement(0)

'*** Get the Batch element
Dim oBatchElem As ACDataElement
oBatchElem = oRTElem.FindChildElementByName("Batch")

'*** Set the Batch Class Name
m_strBatchClassName =
oBatchElem.AttributeValue("BatchClassName")

'*** Get the Pages elem
Dim oPagesElem As ACDataElement
oPagesElem = oBatchElem.FindChildElementByName("Pages")

'*** Keep track of the Page count
Dim lngPageCount As Long = 0

'*** Check if we should check the pages
If IsCheckPagesEnabled() Then

'*** Iterate through each Page element
Dim oPageElem As ACDataElement
For Each oPageElem In _
oPagesElem.FindChildElementsByName("Page")

'*** Increment the Page count
```

```

    lngPageCount = lngPageCount + 1

    '*** Check the page to make sure it should be deleted
    CheckPage(oPageElem, lngPageCount)

    Next oPageElem

End If

End If

End Sub

```

IsCheckPagesEnabled Function

The IsCheckPagesEnabled function determines whether the workflow agent should look for pages marked for deletion through the setup OCX. That is, did the administrator select the "Flag Page for Deletion" option from the Batch Class context menu? This Boolean value is stored in the Batch Class custom storage string.

```

Private Function IsCheckPagesEnabled() As Boolean

Try

'*** Get the BatchClasses element
Dim oBatchClassesElem As ACDataElement
oBatchClassesElem =
m_oSetupDataElement.FindChildElementByName("BatchClasses")

'*** Get the BatchClass element
Dim oBatchClassElem As ACDataElement
oBatchClassElem =
oBatchClassesElem.FindChildElementByAttribute("BatchClass", _
"Name", m_strBatchClassName)

'*** Get the BatchClassCustomStorageStrings
Dim oBatchClassCSSs As ACDataElement
oBatchClassCSSs =
oBatchClassElem.FindChildElementByName _
("BatchClassCustomStorage Strings")

'*** Get the CheckEvenPageDelete custom storage string
Dim oCheckEvenPageCSS As ACDataElement
oCheckEvenPageCSS =
oBatchClassCSSs.FindChildElementByAttribute _
("BatchClassCustomStorageString", "Name", _
"CheckEvenPageDelete")

'*** Make sure we have a reference
If Not oCheckEvenPageCSS Is Nothing Then

'*** Check the value
Return (oCheckEvenPageCSS.AttributeValue("Value") = "True")

End If

Catch ex As Exception

'*** Just return false
Return False

End Try

```

```
End Function
```

CheckPage Subroutine

The CheckPage subroutine checks if a page is odd-numbered and marked for deletion. If so, the value is reset to "False," and a page note is added to indicate that the page should not be deleted because it is an odd-numbered page.

Compile the entire workflow agent code and register it as instructed in the next section.

```
Private Sub CheckPage(ByRef oPageElem As ACDataElement, _
ByVal lngPageNumber As Long)

'*** Check if this is an odd page
If (lngPageNumber Mod 2) = 1 Then

'*** Check if we have the DeletePage CSS
Dim oDeletePageCSS As ACDataElement
oDeletePageCSS = oPageElem.FindChildElementByName _
("PageCustomStorageStrings").FindChildElementByAttribute _
("PageCustomStorageString", "Name", "DeletePage")

'*** Make sure we have a reference
If Not oDeletePageCSS Is Nothing Then

'*** We don't want to delete odd pages. Clear the value.
oDeletePageCSS.AttributeValue("Value") = "False"

'*** Set the Page Note to say that we reset the flag
oPageElem.AttributeValue("Note") = "The DeletePage flag _
was reset...you're not supposed to do that!"

End If

End If

End Sub
```

Registration File Creation

A workflow agent must be registered with Tungsten Capture before it can be used. Once registered, the workflow agent is recognized as valid for a processing queue and can be attached to any Tungsten Capture batch class.

Workflow agent registration is a two-part process that requires you to do the following:

- Set up a workflow agent registration (.aex) file that defines the property settings for the workflow agent.
- Save the registration file and the optional setup OCX to <Tungsten Capture installation folder>\Bin.
- Register the workflow agent through one of the following:
 - Administration module
 - Tungsten Capture Extension Registration Utility

If you are customizing Tungsten Capture, you must have Administrator privileges to install files to the Tungsten Capture installation folder.

Registration File Format

The format for the registration (.aex) file is similar to that of a standard Windows .ini file.

If you have a workflow agent that uses a setup OCX, you must put the setup OCX information in the same registration file as the extension that uses it. The following is the registration file (DeletePageWFA.aex) for the sample workflow agent program.

```
[Workflow Agents]
Delete Page Workflow Agent

[Delete Page Workflow Agent]
WorkflowAgentID=KPSG.DeletePageWFA
WorkflowAgentProgID=DeletePageWFA.Agent
WorkflowAgentFile=DeletePageWFA.dll
Description=This workflow agent makes sure that odd pages are not deleted.
Version=8.0
SupportsNonImageFiles=True
SetupProgram=Delete Even Page Setup

[Delete Even Page Setup]
OCXFile=DeleteEvenPageSetup.dll
ProgID=DeleteEvenPageSetup.SetupControl
Visible=0
```

The registration file (DeletePageWFA.aex) contains:

- A list of workflow agents (identified under the heading [Workflow Agents])
- The workflow agent name (identified under the heading [Delete Page Workflow Agent]) and the following information
 - Workflow ID
 - ProgID of the workflow agent
 - Library (.dll) file name
- The setup OCX for the workflow agent (identified under the heading [Delete Even Page Setup]) and the following information:
 - Setup OCX file name
 - ProgID of the setup OCX

For more information about the registration file, read [Registration File Creation](#).

Registering the Workflow Agent from the Administration Module

1. In the Administration module, on the **Tools** tab, in the **System** group, click **Workflow Agents**. The Workflow Agent Manager window appears.
2. On the Workflow Agent Manager window, click **Add**.
3. From the Open window, browse to the *<Tungsten Capture installation folder>\Bin* and select the .aex file associated with the workflow agent to register.
4. Click **Open**. The workflow agent listed in the Workflow Agents section of the .aex file appears in the **Workflow Agents** window.
5. Select the name of the workflow agent to register and click **Install**. A confirmation message appears when the registration is successful.

6. Click **OK** to clear the confirmation message, and then click **Close** to exit the **Workflow Agent Manager** window.

The newly registered workflow agent is also added to the list of workflow agents available on the Workflow Agents tab of the Create Batch Class and Batch Class Properties windows.

Registering the Setup OCX

Having a setup OCX for your workflow agent is optional; however, if a setup OCX is implemented, it must be registered before it can be used.

Refer to [Registration File Creation](#) for details about creating the registration (.aex) file and using the Tungsten Capture Extension Registration utility.

i If you are customizing Tungsten Capture, you must have Administrator privileges to install files to the Tungsten Capture installation folder.

Because the sample setup OCX is associated with a workflow agent, the registration file for the workflow agent is also used to register the setup OCX. Note the listing of the OCX file and ProgID in the workflow agent registration file in [Format of the Registration File](#).

1. Copy the setup OCX, custom extension files, and registration (.aex) file to <Tungsten Capture installation folder>\Bin on each workstation that runs the Administration module or the custom extension.
2. Register the setup OCX with the **Custom Module Manager** or **Workflow Agent Manager** available from the Administration module. (Alternatively, you can use the Tungsten Capture Extension Registration Utility.)

Registering a Setup OCX Not Associated with a Custom Extension

1. Copy the registration (.aex) file to <Tungsten Capture installation folder>\Bin.
2. Copy the setup OCX to the location specified by the .aex file.
3. Register the setup OCX with the Tungsten Capture Extension Registration Utility by opening the Command Prompt in elevated mode and executing the following command:

```
RegAscEx /f <registration_file_name.aex>
```

Installing and Registering the Workflow Agent

This section gives instructions for installing and registering a workflow agent. Before registration, make sure that you have copied the appropriate files to <Tungsten Capture installation folder>\Bin.

If you are customizing Tungsten Capture, you must have Administration privileges to install files to the Tungsten Capture installation folder.

1. In the Administration module, on the **Tools** tab, in the **System** group, click **Workflow Agents**.
2. Browse to the .aex file for the custom workflow agent, select the file, and click **Open**.

3. On the **Workflow Agents** window, select the workflow agent and click **Install**.
The "Registration complete" message appears if the registration is successful.
4. Click **OK**.
The workflow agent appears in the **Workflow Agent Manager** window.
5. Click **Close**.
6. On the **Batch Class Properties - Workflow Agent** tab, select a workflow agent and click **Add**.
The workflow agent is added to the **Selected Workflow Agents** list.
7. Click **OK** to save your changes and exit the **Batch Class Properties** window.

Removing a Workflow Agent

Use the Administration module to remove or "unregister" a workflow agent from Tungsten Capture. Before removing the workflow agent, you must ensure that it is not used in any published batch class. Otherwise, Tungsten Capture prevents you from removing it.

1. Start the Administration module.
2. On the **Tools** tab, in the **System** group, click **Workflow Agents**.
The **Workflow Agents** window appears.
3. Select the name of the workflow agent to remove.
4. Click **Remove**.
If the selected workflow agent is not used in any published batch class, it is cleared from the list. The **Workflow Agent Manager** does not allow you to remove a workflow agent that is in use by any batch classes.
5. Click **Close**.

Chapter 6

Setup OCX Creation

You can implement a setup OCX to customize batch class setup options. With a setup OCX, you can define custom tabs, configuration settings, and publish checks.

This chapter describes how to design, create, and register a setup OCX, how a setup OCX is loaded and unloaded with the Administration module, and the behavior of setup OCX panels and tabs.

A sample setup OCX associated with the custom workflow agent described in [Workflow Agent Creation](#) is provided and its functionality described later in this chapter.

Setup OCX Design

Determine whether a setup OCX is necessary for your custom extension. A setup OCX can be associated with a custom extension (such as a custom module or custom workflow agent), or designed without a custom extension:

- You can use a setup OCX with an associated custom module to customize the batch class setup process for batch classes that contain the custom module as part of their workflow. The user interface and commands defined in the setup OCX are available only for batch classes that contain the custom module.
- You can use a setup OCX with an associated workflow agent to create custom user interfaces to configure any runtime parameters that the workflow agent may require. The user interface and commands defined in the setup OCX are available only for batch classes that contain the workflow agent.
- You can use a setup OCX without an associated custom extension to customize the batch class setup process for all batch classes. Items defined in the setup OCX are available for all batch classes.

While it is possible to use the standard configuration settings and publish checks available from the Administration module with a custom extension, it is likely that you will want to use a custom setup OCX.

Writing a Setup OCX

The supported programming languages and development environment for creating a setup OCX are the same as for workflow agents. See the topic [Writing the Runtime Module](#) for the supported development environments.

The sample setup OCX provided in this chapter is written in VB.NET and associated with the sample workflow agent provided in [Workflow Agent Creation](#). The sample setup OCX is part of the comprehensive sample for this guide.

The sample setup OCX provides the configuration properties for the workflow agent, and it is available through a context menu for a batch class in the Administration module. Details about the sample setup OCX are provided later in this chapter in [Sample Setup OCX for the Custom Workflow Agent](#).

Code Project Settings

To generate the VB.NET project for the setup OCX, ensure that the following library is referenced in your Visual Basic project: Tungsten Capture Administration Module .NET Type Library (Kofax.Capture.AdminModule.dll).

Sample Setup OCX for the Custom Workflow Agent

The sample setup OCX is part of the comprehensive custom extension sample. This setup OCX for the custom workflow agent, which is described in [Workflow Agent Creation](#), enables the administrator to determine whether a check for even-numbered page deletion is to be enforced and to configure that option through the Batch Class context menu.

Setup OCX for the Workflow Agent

The workflow agent setup OCX (DeleteEvenPageSetup) is written in VB.NET. This custom setup OCX consists of the following code:

- SetupControl is the control code that handles Tungsten Capture events and triggers the setup OCX and setup window.
- SetupForm is the code for the Delete Even Page Setup window that appears when you select "Delete Even Page Setup" from the batch class context menu. The administrator uses this code to enable the workflow agent to check for even-numbered page deletion.

Right-click the batch class in the Batch tree view tab to view the context menu associated with the workflow agent setup OCX.

When you select DeleteEvenPage from the batch class context menu, a Delete Even Page Setup window appears. The setup OCX generates the context menu.

The code segments are listed here:

SetupControl

```
Imports Kofax.SDK.CaptureInfo
Imports Kofax.Capture.AdminModule.InteropServices
Imports Kofax.Capture.AdminModule
Imports System.Runtime.InteropServices
<ProgId("VBNET.SetUpControl")> _
<Guid("08FB952B-69AE-3A90-88C4-268EB2372DD0")> _
Public Class SetUpControl
    Inherits System.Windows.Forms.UserControl
```

```

#Region "Windows Form Designer generated code "
...
#End Region

Private Const cm_strDEPSetup As String = "DEPSetup"
Private Const cm_strDEPSetupText As String = "Delete Even Page Setup"

'*** Reference to the AdminApplication object
Private m_oAdminApplication As InteropServices.AdminApplication

Public WriteOnly Property Application() As InteropServices.AdminApplication
Set(ByVal Value As InteropServices.AdminApplication)
    '*** Cache the object
    m_oAdminApplication = Value
    '*** Initialize the menu
    InitializeMenu()
End Set
End Property

'*****
'*** Function:  ActionEvent
'*** Purpose:  Receive each action event
'*** Input:    nEventNumber - number assigned to event
'*** Output:   vArgument - currently NOT used
'***          pnCancel - Response to the event.
'*** Return:   None.
'*****
Public Sub ActionEvent(ByVal nEventNumber As Integer, ByRef oArgument As Object, ByRef
nCancel As Integer)
    '*** Check the event number to see if the application is about to exit,
'*** if so, clean up m_oAdminApplication's unmanaged resources
    If nEventNumber = InteropServices.KfxOcxEvent.KfxOcxEventLoggedOut Then
        Using oAdminWrapper As New ApiObjectWrapper(Of InteropServices.AdminApplication)
(m_oAdminApplication)
            Debug.WriteLine("Cleans up unmanaged resources handled by m_oAdminApplication")
        End Using
        m_oApp = Nothing
    End If

    '*** Check the event number
    If nEventNumber = InteropServices.KfxOcxEvent.KfxOcxEventMenuClicked Then
        '*** Check the menu text
        If CStr(oArgument) = cm_strDEPSetup Then
            '*** Wraps m_oAdminApplication.ActiveBatchClass property in an instance of
'*** ApiObjectWrapper, so that unmanaged objects used by
'*** m_oAdminApplication.ActiveBatchClass property are released properly
            Using oActiveBatchClassWrapper As New ApiObjectWrapper(Of
InteropServices.BatchClass)(m_oAdminApplication.ActiveBatchClass)
                If Not oActiveBatchClassWrapper.IsNull Then
                    '*** Show the Setup Dialog
                    Dim oSetupForm As LoginForm = New SetupForm
                    '*** Call the Setup Function with the wrapped of ActiveBatchClass
                    oSetupForm.ShowSetupDialog(oActiveBatchClassWrapper.WrappedObject)
                    '*** Unload the Form
                    oSetupForm = Nothing
                End If
            End Using ' Releasing unmanaged resources used by
m_oAdminApplication.ActiveBatchClass
        End If
    End If
End Sub

Private Sub InitializeMenu()
    ' Adds the Delete Even Page Setup to the batch class context menu

```

```

    m_oAdminApplication.AddMenu(cm_strDEPSetup, cm_strDEPSetupText, "BatchClass")
End Sub
End Class

```

SetupForm

```

Imports Kofax.Capture.AdminModule.InteropServices
Imports Kofax.Capture.AdminModule
Imports System.Runtime.InteropServices

Public Class SetupForm

    Private Const cm_strCheckEvenPageDeleteCSS As String = "CheckEvenPageDelete"

    Private m_oBatchClass As InteropServices.BatchClass

#Region "Windows Form Designer generated code"
...
#End Region

    Public Sub ShowSetupDialog(ByRef oBatchClass As InteropServices.BatchClass)
        '*** Cache the reference
        m_oBatchClass = oBatchClass
        '*** Initialize the form
        chkCheckPages.Checked = IsCheckEvenPageEnabled()
        '*** Show the dialog (modal)
        Me.ShowDialog()
        '*** Save the settings
        SetEvenPageEnabled(chkCheckPages.Checked)
    End Sub

    Private Function IsCheckEvenPageEnabled() As Boolean
        '*** Check the BatchClassCustomStorageString
        Dim strBCCSS As String = GetBatchClassCSSSafe(cm_strCheckEvenPageDeleteCSS)
        Return "True".Equals(strBCCSS, StringComparison.OrdinalIgnoreCase)
    End Function

    Private Sub SetEvenPageEnabled(ByVal blnEnabled As Boolean)
        '*** Set the value
        m_oBatchClass.CustomStorageString(cm_strCheckEvenPageDeleteCSS) = IIf(blnEnabled,
        "True", "False")
    End Sub

    Private Function GetBatchClassCSSSafe(ByVal strName As String) As String
        Dim strBatchClassCSS As String = String.Empty
        Try
            strBatchClassCSS = m_oBatchClass.CustomStorageString(strName)
        Catch ex As Exception
            strBatchClassCSS = ""
        End Try
        Return strBatchClassCSS
    End Function

    Private Sub cmdOK_Click() Handles cmdOK.Click
        '*** Simply hide the form
        Me.Hide()
    End Sub
End Class

```

Registering the Setup OCX

Having a setup OCX for your workflow agent is optional; however, if a setup OCX is implemented, it must be registered before it can be used.

Refer to [Registration File Creation](#) for details about creating the registration (.aex) file and using the Tungsten Capture Extension Registration utility.

i If you are customizing Tungsten Capture, you must have Administrator privileges to install files to the Tungsten Capture installation folder.

Because the sample setup OCX is associated with a workflow agent, the registration file for the workflow agent is also used to register the setup OCX. Note the listing of the OCX file and ProgID in the workflow agent registration file in [Format of the Registration File](#).

1. Copy the setup OCX, custom extension files, and registration (.aex) file to <Tungsten Capture installation folder>\Bin on each workstation that runs the Administration module or the custom extension.
2. Register the setup OCX with the **Custom Module Manager** or **Workflow Agent Manager** available from the Administration module. (Alternatively, you can use the Tungsten Capture Extension Registration Utility.)

Registering a Setup OCX Not Associated with a Custom Extension

1. Copy the registration (.aex) file to <Tungsten Capture installation folder>\Bin.
2. Copy the setup OCX to the location specified by the .aex file.
3. Register the setup OCX with the Tungsten Capture Extension Registration Utility by opening the Command Prompt in elevated mode and executing the following command:

```
RegAscEx /f <registration_file_name.aex>
```

Setup OCX Registry Entries

After a setup OCX is installed on a client computer, registry keys must be created on that computer to inform the Administration module how the OCX is loaded, and what tabs should be created. Registry keys that define the Administration module setup OCX are automatically created under the following path during the registration process:

```
HKEY_LOCAL_MACHINE\Software\Kofax Image Products\Ascent Capture\3.0\Ascent  
Capture - Administration\User Panels\
```

The setup OCX registry key values are listed in the following table.

Value	Required	Type	Description
DisplayName	Yes	String	The display name of the panel. This name is displayed on the View tab in the Panels group, and it also appears when the panel is undocked. The Visible value determines whether or not the DisplayName appears.
InitSizeX	No	DWORD	The initial horizontal size of the panel in screen resolution pixels. The default is 50.
InitSizeY	No	DWORD	The initial vertical size of the panel in screen resolution pixels. The default is 50.
MinSizeX	No	DWORD	Minimum horizontal size of the panel when undocked in screen resolution pixels. The default is 50.
MinSizeY	No	DWORD	Minimum vertical size of the panel when undocked in screen resolution pixels. The default is 50.
ProgID	Yes	String	COM ProgID of the OCX.
Type	Yes	DWORD	0 (zero) indicates the setup OCX is used by a custom extension; 1 indicates the setup OCX is not used by a custom extension. The default is 0.
Visible	No	DWORD	1 indicates visible, 0 (zero) indicates not visible. If set to 0, the panel will not be initially displayed and the DisplayName does not appear. The default is 1.

If any of the required registry values are omitted when the Administration module attempts to load a setup OCX, the following error is reported:

```
User defined OCX {User Defined Key} contains an invalid
registry value for {Value Name}.
```

where:

{User Defined Key} is the registry key under "User Panels" causing the error.

{Value Name} is the name of the key value that was omitted.

The application will shut down in this case.

If any of the DWORD values are out of range, the Administration module reports the following error:

```
{Value Name} for user defined OCX {DisplayName} is out of range.
```

Where:

{Value Name} is the name of the value.

{DisplayName} is the value data for the DisplayName value.

The value is ignored, and program execution will continue.

If a registry key is properly constructed, but the OCX cannot be loaded for some reason, the Administration module will report the following error:

```
Unable to create user defined OCX  
{DisplayName} ({Details}).
```

Where:

{DisplayName} is the value data for the DisplayName value.

{Details} indicates the error number or error description.

Tab Registry Keys

Each panel registry key should have a key entitled "Menus" under it. The keys registered under "Menus" define the tabs for that OCX. Keys defined directly under "Menus" dictate the location of a command. The following key names are valid.

To add a command to a tree node:

- BatchClass
- DocumentClass
- FieldType
- FormIdZone
- FormType
- IndexGroupMemberZone
- IndexGroupZoneCollection
- IndexZone
- PageLevelBarcode
- PageLevelBarcodeCollection
- RegistrationZone
- RegistrationZoneCollection
- SamplePage
- SeparationZone

To add a tab to the Ribbon:

- MenuBar

Commands are created by adding keys under the keys that are listed above. The name of the key is the internal name for that command. Also, you can specify the value listed in the table.

Tab Registry Key Value

Value	Required	Type	Description
Text	Yes	String	Display name of the command

If this value is omitted, the command is not added.

The MenuBar key is a special case, such that it requires the value listed in the preceding table. If this value is omitted, the tab does not appear.

MenuBar Registry Key Value

Value	Required	Type	Description
Text	Yes	String	Display name of the tab

Loading the Setup OCX

A setup OCX is loaded with the Administration module as follows:

- **If the setup OCX is associated with a custom extension**, the OCX is loaded:
 - When the Administration module is launched, and if the setup OCX has been registered.
 - While the Administration module is running, if the setup OCX is registered from the Administration module. If the setup OCX is registered with the Tungsten Capture Extension Registration utility, the setup OCX will load the next time the Administration module is launched.
- **If the setup OCX is not associated with a custom extension**, the OCX is loaded:
 - When the Administration module is launched, if the setup OCX has been registered. A setup OCX that is not associated with a custom extension must be registered with the Tungsten Capture Extension Registration utility. If you register the setup OCX while the Administration module is running, the setup OCX will load the next time the Administration module is launched.

Refer to [Registration File Creation](#) for more information about registering custom extensions and setup OCXs.

The first time the setup OCX is loaded, it appears undocked with other display characteristics as defined in the registry settings for the OCX. When a setup OCX is loaded again, it assumes the same location, size, docking status, and visibility used in the previous instance of the Administration module. For details, see [Setup OCX Registry Entries](#).

Note that if your setup OCX is associated with a custom extension, and the setup OCX files are not available in the Tungsten Capture Bin folder when the Administration module launches, a message specifying the name of the custom module that must be registered is displayed. You are prompted to open the Custom Module Manager to register the setup OCX.

If this kind of message displays, make certain that your setup OCX is installed to <Tungsten Capture installation folder>\Bin. Then, click **OK** to register the missing setup OCX.

Setup OCXs that are not associated with a custom extension must reside in the location specified in the .aex file used for registration.

Unloading the Setup OCX

A setup OCX is unloaded from the Administration module as follows:

- If the setup OCX is associated with a custom extension, the OCX is unloaded:
 - When the Administration module is shut down
 - While the Administration module is running, if the custom extension is unregistered from within the Administration module
- If the setup OCX is not associated with a custom extension, the OCX is unloaded when the Administration module is shut down

When a setup OCX is unloaded, its panel and all associated tabs are removed from the Administration module.

Setup OCX Panels

At the discretion of the setup OCX developer, the Administration module can display a panel for each setup OCX. This panel may be resized and moved just like any other panel.

Enabling Panels

Setup OCX panels are enabled as follows:

- If the setup OCX is associated with a custom extension, the panel is enabled when both of the following are true:
 - "Batch class" tab on the Definitions panel is active.
 - Current selection is either a batch class with the custom extension in its workflow, or a component of such a batch class.
- If the setup OCX is not associated with a custom extension, the panel is enabled when the Administration module is launched.

When a panel is disabled, the panel itself remains visible, but the user interface elements of the setup OCX are hidden.

Context Menus

A setup OCX may add one or more menu items to the context menus available from the nodes in the Batch class Definitions panel available in the Administration module. All of the custom menu items are grouped together via separator bars.

The menus can be added programmatically, or they can be specified in the `.aex` file required for custom extension registration. Selecting one of these menu items sends an `ActionEvent` to the OCX, identifying the particular menu selected.

See [Registration File Creation](#) for more information about registering custom extensions.

Enabling Context Menu Items

- The behavior of the context menu items defined with a setup OCX are as follows:
 - Batch class tab of the Definitions panel is active.
 - Current selection is either a batch class with the custom extension, or a component of such a batch class.
- If the setup OCX is not associated with a custom extension, the menu items are always enabled.

When setup OCX menu items are disabled, they are grayed.

Ribbon

A setup OCX can add one or more tabs to the Ribbon in the Administration module and define commands for the tab groups. Selecting one of the custom commands sends an `ActionEvent` to the setup OCX, identifying the particular command selected.

Custom Tab Names

Custom tab names defined by a setup OCX must be different from the standard tab names provided on the Ribbon in the Administration module. If you attempt to create a custom tab with the same name as an existing tab, the following error will occur when the setup OCX is loaded:

```
Cannot create {DisplayName} menu for the user-defined queue
setup module {ProgID}.
```

Where:

`{DisplayName}` is the display name of the tab to be added.

`{ProgID}` is the COM prog ID for the OCX.

For this case, the tab is not added, but program execution is continued.

Enabling/Disabling Custom Commands

- **If the setup OCX is associated with a custom extension**, the commands are enabled when both of the following are true:
 - "Batch class" tab on the Definitions panel is active.
 - Current selection is either a batch class with the custom extension, or a component of such a batch class.
- **If the setup OCX is not associated with a custom extension**, the commands are always enabled.

As an example, a custom tab can be added to the Administration module Ribbon. For this case, the setup OCX that defines the custom tab is associated with a custom extension, and the batch class that contains the custom extension is selected from the Batch class tab of the Definitions panel. The custom tab is then enabled.

When the batch class that contains the custom extension is not selected, the tab name "Sample Index" is available, but the commands associated with it are unavailable.

Panels

Each panel created by a setup OCX will be listed on the tab that appears when you select the View tab and look in the Panels group. Selecting a panel will show or hide the panel, depending on its current state.

 A panel being shown or hidden has nothing to do with it being enabled or disabled. It is possible to show a panel that is disabled. When a disabled panel is shown, the panel elements are not displayed.

It is possible to specify in the registry settings that a panel will always be hidden (see "Visible" key value in the table that appears in [Setup OCX Registry Entries](#)). In that case, that panel will always be omitted from the Panels group.

Batch Class Publishing

A setup OCX can define publish checks to be used by the Administration module when batch classes are published. Note the following:

- **If the setup OCX is associated with a custom extension**, the publish checks defined by the OCX are used when batch classes that contain the custom extension are published.
- **If the setup OCX is not associated with a custom extension**, the publish checks are used when any batch class is published.

No new user elements for publishing can be exposed with a setup OCX; however, changes to the publishing functionality can be defined. When a batch class is published, each queue performs its own publish checks, according to the queue order listed in the workflow. For example, if the

workflow includes a custom extension that uses custom property settings, they are published along with the other property settings for the batch class. Any warnings associated with the custom extension are displayed in the Results list when you publish the batch class.

If all publish checks succeed, the batch class is published. When a batch class is published, the custom properties for each custom extension are also published. If you change any custom properties, they are not reflected in the batch class until you publish it again.

Setup OCX Development API

The complete API for the Tungsten Capture Administration Module .NET Type Library, including properties and methods, is documented in the *Tungsten Capture API Reference*. The properties and methods are associated with setup OCX applications. All properties and methods are defined using Visual Basic syntax.

1. On your Tungsten Capture installation media, navigate to the following location:

`Documentation\Help\APIRef`

2. Double-click **APIRef.chm**.

The *Tungsten Capture API Reference* appears in a browser window.

3. On the Contents tab, click **Kofax.Capture.AdminModule Namespace**.

Chapter 7

Custom Panels and Applications

With the Tungsten Capture Module Type Library, you can add the following custom elements to the Scan, Quality Control, Validation, and/or Verification modules:

- Custom panels. You can add up to 20 custom panels to a standard Tungsten Capture module. Each panel must be developed as an OCX. You can use different panels for each module to be customized.
- Custom commands. In conjunction with a custom panel, you can implement custom commands.

In addition, by taking advantage of the extended features of the Tungsten Capture Module Type Library, you can create standalone custom applications that act as input scripts. This feature (Import Controller) supports virtually any type of custom import application you might care to create.

These custom applications can be run from command lines and can also have their own interfaces that are entirely separate from Tungsten Capture. These choices are entirely up to you.

This chapter describes the following:

- User interface design and behavior
- Custom applications

A sample custom panel associated with the custom module is described in [Custom Module Creation](#) later in this guide.

Programming in a High Availability Environment

When customizing Tungsten Capture in a high availability environment, you should observe the error handling guidelines in [High Availability Environments](#). The guidelines help ensure that your applications take full advantage of Tungsten Capture's high availability features.

For more information on high availability, refer to the *Tungsten Capture Installation Guide*.

User Interface Design and Behavior

The following sections describe the user interface design and behavior of custom panels and commands, which may be added to the Scan, Quality Control, Validation, and/or Verification modules. Each panel contains a user-defined OCX.

New or modified customizations can take advantage of the Fluent UI in the Tungsten Capture modules on the preceding list.

Custom Panels

As an example, for a custom panel in the Validation module, the user OCX is contained completely within a new user interface panel. This panel may be resized and moved just like any standard panel. The OCX panel sends resizing events to the child OCX whenever the panel itself is resized.

The OCX may display any desired graphical objects on the panel.

The outside edge of the panel (about 4 pixels) is used for panel operations (listed below). Any panel clicks outside this margin area are passed to the OCX. If the OCX obtains focus, then keyboard events are also sent to it.

Standard panels and new OCX panels support the following operations:

- Saving the panel size, state (floating/docked), and position between application invocations.
- Clicking and dragging the top edge of a panel causes the panel to move to a new location on the frame. If the panel is dragged off the frame, it becomes a distinct, floating window.
- Double-clicking the top edge of a docked panel changes that panel into a floating window. (Double-clicking the top edge of the panel toggles between a docked/undocked state.)
- Resizing a floating window. The panel has a minimum size (which is 50,50 for default panels).
- Hiding a floating window by clicking its Close box. The panel can be restored by selecting it from the View tab in the Panels group.
- Dragging a floating window back to the frame to redock the panel to the frame.

 The OCX cannot programmatically resize its parent window.

A panel may receive several resize events just after creation. An initial resize of size 0,0 occurs when the OCX is first built. Windows may send other events, depending on the screen resolution configuration.

Any changes to data made by the OCX (field content, field data, reject flag, note information) are reflected by all standard Tungsten Capture controls, including the image viewer, the tree view, the data entry panel, and the thumbnail view.

Note the following:

- Panels without windows: A "visible" flag in the registry determines if a panel is displayed in the Panel group on the View tab. If a panel is not visible, it never appears in the Panel group. The OCX has an API called ShowWindow(), which allows it to display itself.
If the panel has the visible flag = 0 in the registry, it does not appear in the Panel group. It can still be displayed via command selections. If the panel is visible when the application closes, it is visible when the application opens.
As a workaround, don't display the panel. Instead, the OCX can display a modal window. Or, you can hide the panel after opening the application.
- Panels hidden by the OCX: The application API allows the panel to hide/show itself.

Custom Panels in the Fluent User Interface

Updated or new custom panels may take advantage of the Fluent User Interface (UI). The method `AddMenuEx` is provided explicitly for this purpose. This method functions exactly like the older `AddMenu` method, but with several new parameters geared toward the Fluent UI.

See the *Tungsten Capture API Reference* for details.

Themes in the Fluent User Interface

The look and feel for legacy custom panels may not conform to the look and feel of Tungsten Capture with the Fluent UI. By default, the panels appear in their native Windows style. The only way to provide a matching look and feel is to change the Tungsten Capture theme to match the style of the custom panel.

An event and APIs have been provided to allow modified or new customizations to work properly with themes.

Custom Tabs

The View tab contains a Panels group that allows hiding/showing the user panel. The User Panel Name in the group is configurable.

In addition, each OCX can add a tab to the Ribbon. The name of the new tab must be different from any existing tab text. Duplicates are not added. If you attempt to add duplicate text, an error is generated.

The Ribbon text and tab items can be read from the registry. The registry supports one Ribbon per panel. Multiple Ribbons may be defined programmatically.

When more than one OCX (user-defined panel) exists in the registry, they are added in the order they are loaded. The registry alphabetizes its entries. If added programmatically, the tabs are inserted in the order that the panels are loaded.

Commands such as Show Me, Hide Me, and Show window can be defined as part of your OCX, as shown in the sample OCX provided in your installation folder. See [Sample Custom Panel](#) for more information about installing and registering the samples.

Scan, Quality Control, Validation, and Verification Tree Node (Context) Menus

Each OCX may add one or more items to the tree nodes displayed in the Batch Contents panel. The menu text, accelerator, and tree nodes are specified in the registry or you can add them programmatically. Selecting one of these menu items sends an action event to the OCX. The action event identifies the menu item selected.

 The OCX developer is responsible for accelerator key uniqueness.

Tabs Can Be Added, Removed, and Edited at Runtime

The AddMenu() API allows the addition of commands by the OCX while loading or running. A ShowMenu() API allows a user-defined command to be shown or hidden. Refer to the *Tungsten Capture API Reference* for details about AddMenu() and ShowMenu().

Registry Entries for Tabs

Each user-defined panel has a Menu key under it. The Menu key has subkeys for each tab that allows the insertion of commands. Each tab has a name and properties. Currently, the only property is the tab text.

The table describes the Menu registry keys.

Registry Keys

Key Entries	Description
Menu	This key indicates that tabs will follow. The locations are under the tabs.
Batch Document MenuBar Page	Locations for tabs to appear. The following sample code shows the tab location names. <pre> *** Possible locations for user-defined *** menus for Scan, Quality Control, *** Validation, and Verification m_RuntimeMenuLocationKeys(0) = "MenuBar" m_RuntimeMenuLocationKeys(1) = "Batch" m_RuntimeMenuLocationKeys(2) = "Document" m_RuntimeMenuLocationKeys(3) = "Page" </pre> The Ribbon contains one additional text key from the Ribbon to identify the text displayed on the frame.
Batch Menu Sel	Tab event text, supplied by the developer. This is the text to be returned to the OCX when the command is selected. This text is passed back as part of the MenuClicked() action event.
Text "&Pick me..."	"Text" is the non-localized value that contains the text to be displayed. This contains the accelerator key ("&" on the "P").

For example, MenuBar: "&Sample OCX" is the Ribbon text added from a registry entry. A tab entry "&OCXTest" was added by calling AddMenu() within the OCX.

The MenuBar entry contains one string value; it is the tab text for the Ribbon with the accelerator. You can programmatically add multiple tabs using AddMenu().

 The OCX developer is responsible for keytip uniqueness.

Custom Panel Installation

To install a custom panel, you must copy and register the OCX on each client computer in any folder location. Also, you must install any required customer DLLs (if applicable).

i Custom panels are not included as part of the standard Tungsten Capture installation. However, sample custom panels are available in your Tungsten Capture installation folder (a VB.NET sample is located in `Source\Sample Projects\StdCust`). For more information, see [Sample Custom Panel](#).

Next, some registry entries (on each client computer) must be created under the following path:

```
HKEY_LOCAL_MACHINE\Software\Kofax Image Products\Ascent Capture\3.0\{module name}\User_Panels\{User-Defined-Key}\
```

The "local machine" area is utilized since such a configuration is more likely to be based on the computer than on the user. The `{<module name>}` must be replaced with one of the following names:

Ascent Capture – Scan

Ascent Capture – Quality Control

Ascent Capture – Validation

Ascent Capture – Verification

Ascent Capture – Administration

Although you may be working with a more recent Tungsten Capture installation, notice that the registry entry is created under a path that intentionally includes a reference to "3.0."

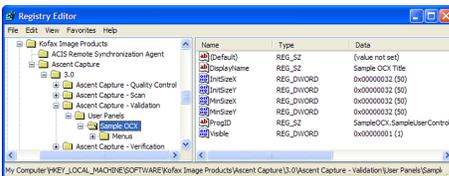
Within the User Panels path, the `{User-defined Key}` key is any valid registry key. If more than 20 keys exist in this folder, then subsequent keys are ignored.

Value	Required?	Value Type	Notes
DisplayName	Yes	String	The name may appear in the title bar of the undocked panel or in the Panels group. The Panels group is available on the View tab. Currently, this tab contains the name of a user-defined OCX and the end user is allowed to turn the panel on and off. The OCX has no control over this functionality. The Visible value setting determines whether or not the panel name appears in the Panels group.
InitSizeX	No	DWORD	The initial horizontal OCX size in screen resolution pixels. The default is 50.

Value	Required?	Value Type	Notes
InitSizeY	No	DWORD	The initial vertical OCX size in screen resolution pixels. The default is 50.
MinSizeX ¹	No	DWORD	The minimum undocked horizontal OCX size in screen resolution pixels. The default is 50.
MinSizeY ¹	No	DWORD	The minimum undocked vertical OCX size in screen resolution pixels. The default is 50.
ProgID	Yes	String	The COM program ID of the OCX.
ReplacesIndexFieldsPanel	No	DWORD	<p>This applies to the Validation and Verification modules only.</p> <p>When enabled, the Index Fields panel and its View, Toolbars, and Index Fields commands are hidden. Additionally, its context menu does not show the Index Fields menu item.</p> <p>When an eDocument is displayed, the internal Windows Explorer panel takes focus after displaying the eDocument. If the OCX has registered itself as ReplacesIndexFieldsPanel, it will receive a new event (KfxOcxEventResetFocus) indicating that focus has been returned to the OCX. In such cases, Tungsten Capture gives focus to the "outer" OCX panel. However, you may want to set focus to some subcontrol. To do this, catch the new event and set focus as desired.</p> <p>Only one OCX can register itself as ReplacesIndexFieldsPanel.</p> <p>Note that the panel works best when it is docked.</p>

Value	Required?	Value Type	Notes
Visible	No	DWORD	Determines if the panel is displayed in the Panels group. If the panel is not visible, it can still be displayed via commands. Defaults to 1 (visible); 0 is not visible.

¹ The minimum size parameters affect only the undocked minimum size. There is no support for minimum docked size.



Registry Values for the {User-Defined Key}

The initial docking location is always at the upper left of the client area.

If any required values are missing, the application displays the following error message and then shuts down:

```
User-defined OCX "{User-defined Key}" contains any invalid registry values for {Key Name}.
```

Because the OCX is an integral part of the application, the user is not allowed to run the application unless the OCX is properly set up. If absolutely necessary, the user may completely remove the OCX registry key and the software will run. However, this is not recommended as invalid data may occur (assuming the OCX is performing the necessary tasks).

If any size parameter is out of range, the following message appears:

```
{parameter} for user-defined OCX {DisplayName} is out of range.
```

The "{parameter}" string is one of the values shown in the preceding table. If all parameters are read and valid, but the OCX cannot be created for some reason, the following message appears:

```
Unable to create user-defined OCX {DisplayName} ({details})
```

Where:

{details} indicates the error number and/or description.

Invoking Tungsten Capture Commands from a Custom OCX Panel

You can use a custom OCX panel to invoke Tungsten Capture commands in the Administration, Scan, Quality Control, Validation, and/or Verification modules through the SelectMenuItem function.

SelectMenuItem Function

Function	Arguments	Description
SelectMenuItem	Returns: Boolean Parameters: <i>lResourceID [ByVal; Long]</i> <i>bSendImmediate [ByVal Optional Default to FALSE]</i>	Sends a selected command to a Tungsten Capture module. Resource ID of command.

If successful, SelectMenuItem returns a nonzero value. In the case of an unexpected system error, it returns a value of zero. SelectMenuItem throws an error if any of the following are true:

- You specify an invalid resource ID for the command
- Item is currently grayed in the target module
- You execute an exit command immediately (bSendImmediate is True)
- Application is busy

The bSendImmediate parameter is optional, and it can be set to True or False.

- If True, bSendImmediate invokes the Windows API call "SendMessage," which executes the command instantly and fails if the module is processing another event.
- If False, bSendImmediate invokes the Windows API call "PostMessage," which posts the command so it is executed when other events are completed.

Passing a Command to the Scan Module

The following sample passes a command from a custom OCX to the Scan module. The command shown here displays the last page in a batch.

```
Private Sub ScanStopped()
    Dim bResult As Boolean = False
    Dim lMenuResourceNumber As Integer =
        InteropServices.KfxApiScanMenuItems.KfxScanMenuPageLast
    Dim bSendImmediate As Boolean = False
    Try
        bResult = m_oApp.SelectMenuItem(lMenuResourceNumber, bSendImmediate)
    Catch ex As Exception
        MessageBox.Show(String.Format("Unable to select menu: {0}", ex.ToString()))
    End Try
End Sub
```

OCX Tab Selection

In the following code, the OCX Tab Selection invokes the selection of About Tungsten Capture in the Administration module.

```
Dim bResult As Boolean = False
Dim bSendImmediate As Boolean = False

'*** Display the about box
bResult = m_oApp.SelectMenuItem(InteropServices.KfxApiAdminMenuItems.
    KfxAdminMenuHelpAboutAscentCapture, bSendImmediate)
```

In the preceding sample, `KfxAdminMenuHelpAboutAscentCapture` is the item for the About Tungsten Capture command in the Administration module. A separate set of commands is available for each Tungsten Capture module.

Sample Custom Panel

A sample custom panel is provided as part of your Tungsten Capture installation. The sample is installed to the following folder:

- `<Tungsten Capture installation folder>\Source\Sample Projects\StdCust`

Two subfolders are included:

- `OCXPanel` (or `OCXPanel.NET`): This folder contains Visual Basic source code for a sample OCX.
- `OCXReg` (or `OCXReg.NET`): This folder contains Visual Basic source code for a sample registration utility.

The source code in the preceding folders is provided in Visual Basic .NET. The sample OCX is provided to give you a sample of a simple custom OCX. The sample registration utility demonstrates the type of utility you might provide to customers for installing and registering your sample OCX for use with Tungsten Capture.

Sample Custom Panel Registration

Custom panel registration is a two-part process, requiring you to do the following:

- Use a utility (such as the sample utility provided in your `OCXReg` folder) to register the custom sample panel with Tungsten Capture.
- For COM components, use `RegSvr32.exe` to register a custom panel in the Windows Registry. For .NET components, use `RegAsm.exe` to register a .NET custom panel for COM Interop with the Windows Registry and the .NET Framework.

The procedure explains how to use the sample registration utility to register the sample OCX panel for use with Tungsten Capture.

Registering the Sample Custom Panel with Tungsten Capture

The procedure explains how to use the sample registration utility to register the sample OCX panel for use with Tungsten Capture.

1. Start Windows Explorer and browse to the following folder:

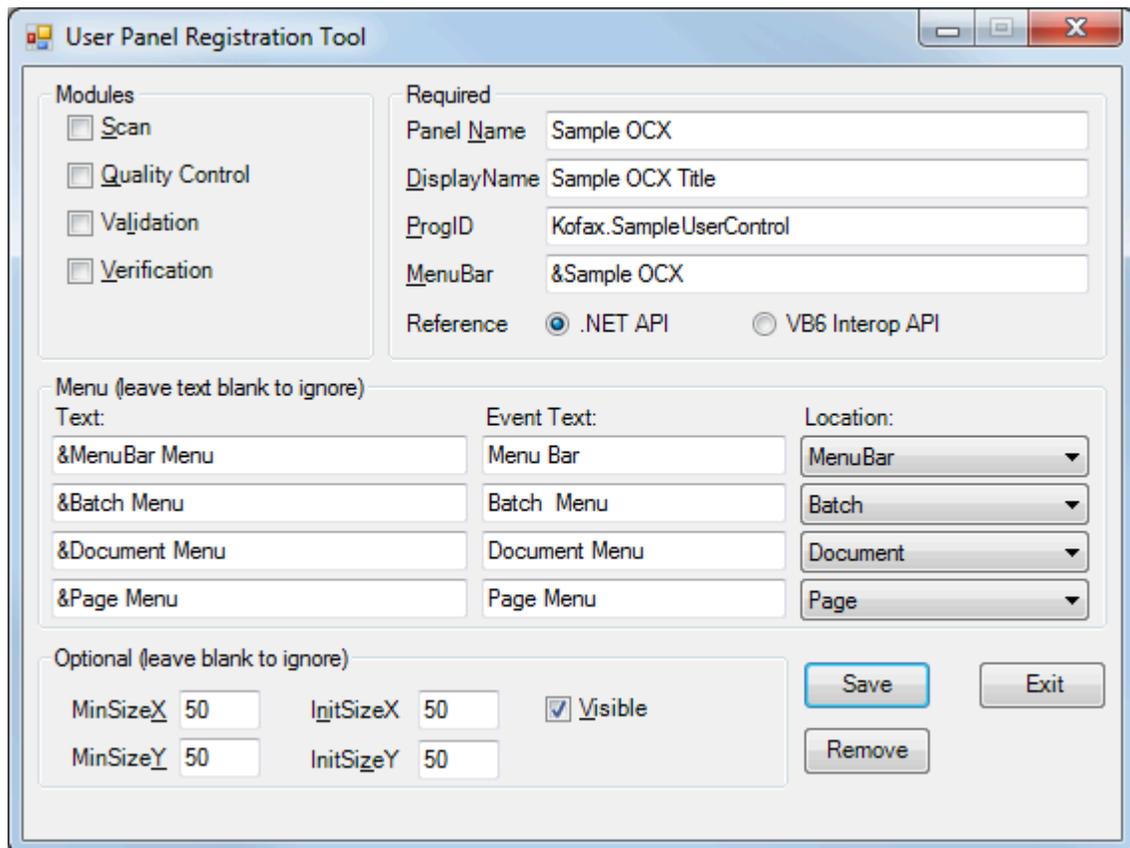
`C:\Program Files (x86)\Kofax\Source\Sample Projects\StdCust\OCXPanel\`

2. Copy `SampleOCX.dll` to the `\Bin` folder.

3. Browse to the following folder:

`C:\Program Files (x86)\Kofax\Source\Sample Projects\StdCust\OCXReg\`

4. Start the User Panel Registration Tool (`PanelReg.exe`), shown in the figure.



Registering a Custom Panel

5. In the **Modules** group, select the modules in which you want to enable the sample OCX panel.
6. In the **Required** group, in the **ProgID** box, replace the default SampleOCX.SampleUserControl with Kofax.SampleUserControl.
7. You do not need to change the other default settings, unless you are working with custom panels carried over from previous versions. In that case, you would select VB6 Interop API as the Reference setting.
8. Click **Save**.

Registering the Sample Custom Panel in the Windows Registry

The procedure explains how to use the sample registration utility to register the sample OCX panel in the Windows Registry.

After the registration process, the custom panel is available in each module for which it was registered with Tungsten Capture. For details about typical custom panel behavior, see [User Interface Design and Behavior](#).

1. On the Windows Start menu, click **Run**.
2. In the Run window, type the following:

```
<path1>RegSvr32.exe
  <path2>Ocxname.ocx
```

Where:

<path1> is the full path to the RegSvr32 application.

<path2> is the full path to Ocurname.ocx.

Ocurname.ocx is the name of your custom panel OCX.

Registering the Sample Custom Panel

1. On the Windows Start menu, click **Run**.
2. In the Run window, browse to the OCXPanel folder included in your Tungsten Capture installation and type the following:

```
<path1>RegSvr32.exe
  <path2>Smplocx.ocx
```

Where:

<path1> is the full path to the RegSvr32 application.

<path2> is the full path to Smplocx.ocx.

Smplocx.ocx is the name of the custom panel OCX provided with Tungsten Capture.

Registering the Sample .NET Custom Panel

1. On the Windows Start menu, click **Run**.
2. In the Run window, browse to the OCXPanel folder included in your Tungsten Capture installation and type the following:

```
<path1>RegAsm.exe
  <path2>OCXname.dll
```

Where:

<path1> is the full path to the latest .NET Framework:

C:\WINDOWS\Microsoft.NET\Framework\<version>\

<path2> is the full path to OCXname.dll.

OCXname.dll is the name of the sample .NET custom panel OCX provided with Tungsten Capture.

Sample Custom Panel in VB.NET

The following sample custom panel named "DelPagePanel" has no user interface but enables an operator to flag a page for deletion. This custom panel uses a context menu accessible from a batch to mark pages for deletion.

When you right-click the page to delete, a context menu appears, and you can use it to mark the page for deletion.

The following is the code for the custom panel (DelPagePanel).

```
Imports Kofax.SDK.CaptureInfo
Imports Kofax.Capture.CaptureModule.InteropServices
Imports Kofax.Capture.CaptureModule
Imports System.Runtime.InteropServices
<ProgId("VBNET.DelPagePanel")>
<Guid("08FB952B-69AE-3A90-88C4-268EB2372DD1")> _
Public Class DelPagePanel
  Inherits System.Windows.Forms.UserControl
```

```

#Region "Windows Form Designer generated code "
...
#End Region

Private Const cm_strDeletePageMenu As String = "DeletePageMenu"
Private Const cm_strDeletePageMenuText As String = "Flag Page for Deletion"
Private Const cm_strPageCSSName As String = "DeletedPage"

'*** Reference to the Application object
Private m_oApplication As InteropServices.Application

Public WriteOnly Property Application() As InteropServices.Application
Set(ByVal Value As InteropServices.Application)
'*** Cache the object
m_oApplication = Value
'*** Initialize the menu
InitializeMenu()
End Set
End Property

'*****
'*** Function: ActionEvent
'*** Purpose: Receive each action event
'*** Input: nEventNumber - number assigned to event
'*** Output: vArgument - currently NOT used
'*** pnCancel - Response to the event.
'*** Return: None.
'*****
Public Sub ActionEvent(ByVal nEventNumber As Integer, ByRef oArgument As Object, ByRef
nCancel As Integer)
'*** Check the event number to see if the application is about to exit,
'*** if so, clean up m_oApplication's unmanaged resources
If nEventNumber = InteropServices.KfxOcxEvent.KfxOcxEventLoggedOut Then
Using oAdminWrapper As New ApiObjectWrapper(Of InteropServices.Application)
(m_oApplication)
Debug.WriteLine("Cleans up unmanaged resources handled by m_oApplication")
End Using
m_oApplication = Nothing
End If

'*** Check the event number
If nEventNumber = InteropServices.KfxOcxEvent.KfxOcxEventMenuClicked Then
'*** Check the menu text to make sure we got the right menu
If CStr(oArgument) = cm_strDeletePageMenu Then
'*** Flag the page for deletion by setting the Page Custom Storage String
'*** First, wraps m_oApplication.ActiveBatch property in an instance of
'*** ApiObjectWrapper, so that unmanaged objects used by
'*** m_oApplication.ActiveBatch property are released properly
Using oActiveBatchWrapper As New ApiObjectWrapper(Of InteropServices.Batch)
(m_oApplication.ActiveBatch)
If Not oActiveBatchWrapper.IsNull Then
'*** Then flag the page for deletion by setting the
'*** Page Custom Storage String
Using oActivePageWrapper As New ApiObjectWrapper(Of InteropServices.Page)
(oActiveBatchWrapper.WrappedObject.ActivePage)
If Not oActivePageWrapper.IsNull Then
oActivePageWrapper.WrappedObject.CustomStorageString(cm_strPageCSSName) = "True"
End If
End Using ' Releasing unmanaged resources used by ActiveBatch.ActivePage
End If
End Using ' Releasing unmanaged resources used by m_oApplication.ActiveBatchClass

```

```
    End If
  End If
End Sub

Private Sub InitializeMenu()
  ' Adds a menu for deleting the page
  m_oApplication.AddMenu(cm_strDeletePageMenu, cm_strDeletePageMenuText, "Page")
End Sub
End Class
```

Chapter 8

Custom Module Creation

You can create a custom module to perform tasks such as document separation, page registration, form identification, automatic or manual indexing, verification, or full text OCR. Custom modules can also perform unique functions suited to your business needs. For example, you can write a custom OCR module that uses an engine other than the engines provided with Tungsten Capture. Or, the custom module might perform a special type of image processing, such as bar code recognition on color images or form identification using a custom algorithm.

This chapter gives an overview of the development process for creating a custom module for Tungsten Capture. A sample custom module, which is part of the comprehensive example of a custom workflow agent, setup OCX, and custom panel, is also provided in this chapter.

Custom Modules

You can implement a custom module alongside the standard set of Tungsten Capture processing modules, or you can use it to replace a standard module. Once you create the custom module, it must be registered for use with Tungsten Capture and added to the batch class workflow (which must include the standard Scan module to introduce a batch into the Tungsten Capture processing environment, as well as the standard Export module as the final processing module).

i The Tungsten Capture installation process may require that an interactive custom module be shut down before installation continues. Therefore, custom modules should properly handle Microsoft Windows close messages. If a custom module does not properly handle close messages sent to terminate its process, Tungsten Capture upgrade installations may fail.

High Availability Environments

When customizing Tungsten Capture in a high availability environment, you should observe the following error handling guidelines. The guidelines help ensure that your applications take full advantage of the Tungsten Capture high availability features.

For more information on high availability, refer to the *Tungsten Capture Installation Guide*.

Error Handling Guidelines

Any component of a highly available system is subject to failure at any time. To meet high availability requirements, design your Tungsten Capture customizations to detect and recover from errors during operations that require access to a remote computer.

These errors can be grouped into the following categories:

- [Database Operations](#)
- [File System Operations](#)
- [Client / Server Operations](#)
- [Third-Party Operations](#)

Database Operations

Any customization that accesses a database on another computer may encounter errors. If the customization accesses a database on a cluster server, the database will be off-line for a short period of time during a failover. Database errors can occur due to a variety of issues including, but not limited to, intermittent network failures and cluster failover.

To ensure high availability, your application must check for errors on every database operation. If an error occurs, the application must reconnect to the database and retry the operation if necessary.

Even if an error occurs, it is possible that the database operation succeeded but a network failure prevented the successful result from reaching your application. Therefore, if it is important that the failed operation be executed exactly once. You must determine if it succeeded or failed to know if the operation should be retried.

Since it takes a period of time for the database to failover, applications should continue to retry for at least 2 minutes.

File System Operations

Any customization that accesses files on another computer may encounter errors. If the customization accesses a file system on a cluster server, then during a failover, the file system is off-line for a short period of time. File access errors can occur due to a variety of issues including, but not limited to, intermittent network failures, contention, and cluster failover.

To ensure high availability, your application must check for errors on every file operation. If an error occurs, it must retry the operation. Note that when an operation returns an error, it is possible that the file operation started, but did not complete. Therefore, when retrying, you must take care to determine if the same parameters can be used or if the failed operation changed the state of the system.

Since it takes a period of time for the file system to failover, applications should continue to retry for at least 2 minutes.

Client / Server Operations

Any customization that accesses a third-party server application on another computer may encounter errors. The client/server application might be "cluster-aware" and be deployed on the cluster server, or it may provide some other mechanism to make the server highly available. In any case, it is up to the client/server application utilized by the Tungsten Capture customization to provide high availability in some way.

Third-Party Operations

Any third-party function utilized by a Tungsten Capture customization might access a remote computer.

To ensure high availability, your application must check for errors after every function call. If an error occurs, the application must determine if a retry is needed.

Sample Applications

A custom module sample named "DeleteEvenPage," which is part of the comprehensive VB.NET example, is used to demonstrate the design and implementation of a custom module. The code for this custom module is provided later in this chapter.

Tungsten Capture also ships with a sample custom module that is provided as part of your Tungsten Capture installation. This Visual Basic .NET program is located here:

```
<Tungsten Capture installation folder>\Source\Sample Projects\CustMod\Generic
```

The folder contains the files necessary to register a custom module named "Sample." The sample includes a generic example of a custom module, which you can install and register for demonstration purposes. For details about the sample custom module, see [Custom Module Sample](#).

Typical Development Tasks

The typical custom extension development process includes the tasks outlined in this section. You may decide to perform some tasks in a different order.

- Design the custom extension
- Configure the setup OCX to store properties and add publish checks
- Write the runtime application (use Tungsten Capture Document Access API library elements to create an application to interact with Tungsten Capture)
- Create the custom extension registration file
- Register the custom extension
- Create the installation program

Design the Custom Module

As with any development project, the success of the custom module implementation process requires careful planning and analysis. Before starting any development, consider how your custom module will augment the standard Tungsten Capture functionality.

For example, be sure to take into account the characteristics of the forms you expect to process with the custom module. Then decide whether to make the custom module available in addition to, or in place of, existing Tungsten Capture functions.

Additionally, you need to determine if it would be appropriate to add custom configuration settings to support the new functionality. Then decide where (tabs or user interface panels) it would be appropriate to make the custom settings available to the user. You should produce specifications that define the design, functionality, and scope for the custom extension.

Create the Setup OCX

You will probably want to develop a setup OCX in the Administration module to store configuration properties and publish checks associated with the custom extension. The setup OCX is optional, because you could potentially use the standard Help for Tungsten Capture settings. For example, if you developed a custom module to replace the standard Validation module, you might be able to use the standard settings and publish checks. However, it is likely that most custom extensions will require unique configuration properties and publish checks. For details, see [Setup OCX Creation](#).

Write the Runtime Application

Use the Tungsten Capture Document Access API library to integrate the application into your Tungsten Capture installation. You can present a list of batches to the user, or you can opt to have the custom module automatically process batches as they become available. Develop the custom module to meet your specifications.

Tungsten Capture Document Access also allows XML batch information to pass between Tungsten Capture and the custom module. As the custom module receives a batch, it performs any custom functions, and then sends the batch back to Tungsten Capture.

i Your XML transport files need to follow the format required for compatibility with Tungsten Capture. Sample Document Type Definition (DTD) files required for custom module XML files are provided with Tungsten Capture and installed with the program.

Create the Custom Module Registration File

You must create a registration file that defines the property settings for the custom module. The file must be in place before you register the custom module.

The following is the registration file (`DeleteEvenPage.aex`) for the sample custom module:

```
[Modules]
Delete Even Page CM

[Delete Even Page CM]
RuntimeProgram=DeleteEvenPage.exe
ModuleID=DeleteEvenPage.exe
Description=This module deletes even pages...a very useful tool
Version=8.0
SupportsTableFields=True
SupportsNonImageFiles=True
```

The registration file (`DeleteEvenPage.aex`) contains the following:

- Name of the custom module (identified under the heading `[Modules]`)
- Custom module name (identified under the heading `[Delete Even Page CM]`)
 - Runtime executable for the custom module

- Module ID
- ProgID of the custom module

For more information about the registration file, read [Registration File Creation](#).

Register the Custom Module

Once the registration file is in place, you register the custom module so that Tungsten Capture will recognize it. After registration, you can also test the setup OCX to verify its validity. For details on registering custom extensions, see [Registration File Creation](#).

After registration, you can add the custom extension to the batch class so you can test it. Once you are satisfied with the preliminary tests, you can proceed to create an installation program.

Create an Installation Program

If you plan to distribute the custom module, you need to write a program to add the application to your Tungsten Capture installation. You can optionally incorporate the registration process into the installation program. You need to install the custom extension setup OCX, custom extension runtime, and registration file to `<Tungsten Capture installation folder>\Bin` for every workstation where you want the custom extension to be available. You can also add a custom extension icon to the Tungsten Capture program group.

Document Routing

As a custom module developer, you can use the Document Routing feature to divide a batch into multiple batches. You do this by opening the parent batch and then creating a child batch. You can move pages and documents between the two open parent and child batches. You can divide batches based on any programmable criteria such as form type.

Document Routing Functions

The batch create, move, and close operations are performed by the following functions:

- [ChildBatchCreate \(DBLite\)](#)
- [MoveElementToBatch \(DBLiteOpt\)](#)
- [BatchCloseWithModuleID \(DBLite\)](#)

ChildBatchCreate

ChildBatchCreate creates a child batch based on an existing open parent batch. Each child batch uses the same published batch class as the parent batch. Batch classes are copied as well.

The following portion of code taken from the CMSplit sample custom module creates a child batch based on a parent batch which, in this example, is passed to the SplitDocuments subroutine.

```
Private Sub SplitDocuments ( _  
    ByVal oBatch As Kofax.Capture.SDK.CustomModule.IBatch, _  
    ByVal oList As List(Of Kofax.Capture.SDK.Data.IACDataElement) )
```

```

If oList IsNot Nothing AndAlso oList.Count > 0 Then
    '*** Create the child batch
    Dim oChildBatch As Kofax.Capture.SDK.CustomModule.IBatch = oBatch.ChildBatchCreate()
...
End If
End Sub

```

Only a single child batch can be open at the same time as a parent batch. A custom module must close a child batch before it can create another one from the same parent. If your module closes both parent and child batches, it can then reopen the child batch, making it a parent, and then create a child batch based on it.

Naming a Child Batch

When your custom module creates a child batch, you can add code that names the batch. If you do not provide naming code, the default child batch name is the parent batch name appended with the date and time.

Closing Batches

A custom module cannot close a parent batch until it first closes the child batch, with the following exception: If you exit a custom module, causing the module to close, Tungsten Capture suspends both parent and child batches.

If a batch fails to close, it throws an error, allowing you to add appropriate error handling. If, for example, a custom module passes valid closing parameters to a function, and then fails to close, your error handling code can attempt to close the batch again. For more information about error codes, see the *Tungsten Capture API Reference*.

MoveElementToBatch

MoveElementToBatch moves an ACDataElement (page or document) from one open batch to another. MoveElementToBatch automatically deletes the ACDataElement from the source batch.

The following code moves a document from oDocElement to oChildDocsElement (both of type ACDataElement).

```

'*** Split this current document into the child batch
oChildDocsElement.MoveElementToBatch(oDocElement)

```

MoveElementToBatch returns a new object from the destination batch containing the object that was moved.

Also, note the following:

- A custom module cannot move an element (document or page) to a specific location in the destination batch. If necessary, a custom module can modify an element after it has been moved.
- If the source page is the last page in a document or batch, you must delete the parent document or batch. Similarly, if the source document is the last document, you must delete the parent batch.

i When your custom module moves pages or documents using the Document Routing feature, there is no licensing charge.

BatchCloseWithModuleID

BatchCloseWithModuleID closes and unlocks a batch to be routed to a specified module.

BatchCloseWithModuleID Parameters

Input Parameter	Description
eNewState	Specifies the new state of the batch once it is unlocked.
strModuleID	A unique ID (such as ocr.exe) that specifies the module where the batch is routed.

About Document Routing Features

Document Routing includes the features described in this section.

Document GUID

When a custom module moves a document, its DocumentGUID is not changed, but its Page ID is changed.

Tracking Statistics

As a custom module moves pages or documents from a parent batch to a child batch, user tracking statistics data reflects the page movement from both the source and destination batch.

Reference Batch ID

When a batch is created, the Tungsten Capture system assigns it a Reference Batch ID. The Reference Batch ID is a GUID for each originally created batch. However, it has the same value for all batches created by calling CreateChildBatch from a parent batch.

This value is used to track the original batch and all child batches through the system. You can use this value as a batch field, index field (for a document or folder), export, or PDF header value.

Tungsten Capture also adds the Reference Batch ID as a column in the StatsBatch table.

You can also use batch-specific Tungsten Capture values as endorser values. In this case, the Reference Batch ID is available, but it is not expanded.

Unsupported Features

The Document Routing feature does not support:

- Batch totals
- Partial batch export
- Folders

Using Tungsten Transformation

You can use the Document Routing feature as follows:

- In your own custom module, independent of Tungsten Transformation.
- Using only the Tungsten Transformation implementation of Document Routing.

i Tungsten Transformation stores data in its own external files. Using your own custom module in conjunction with the Tungsten Transformation implementation may have unpredictable results.

Using the Sample Custom Module

This section explains how to use the provided sample custom module (CMSplit), which demonstrates the use of the Document Routing feature.

The sample custom module divides a batch based on form type. It creates new batches for each form type except for the first form type identified. For example, if a batch contains scanned documents based on three form types, the sample custom module divides the existing batch into two additional batches. The original batch contains documents based only on the first identified form type. If the original batch contains only one page or document, no child batches are created.

In the workflow, the sample custom module is typically placed after the Recognition Server module.

The sample custom module contains a single-form user interface that displays status only as it processes batches (similar to the Separate sample custom module).

The sample custom module is located here:

```
<Tungsten Capture Installation Folder>\Capture\Sample Projects\CustMod\CMSplit
```

CMSplit is written in VB.NET using .NET 4.0. The CMSplit sample shares code with the Tungsten Capture Separation Module sample (separate.exe).

Sample Custom Module

The sample program named DeleteEvenPage is an unattended module that is designed to run after the Scan module. It checks that the pages marked for deletion are removed from the batch.

The following is the code for the custom module named DeleteEvenPage. This class definition makes up the blueprint for the custom module. In the Initialize function, we log into Tungsten Capture and cache the appropriate data structures.

DeleteEvenPage Module

```
Imports Kofax.Capture.DBLite
Imports Kofax.Capture.SDK.CustomModule
Imports Kofax.Capture.SDK.Data

Public Class Driver
    '*** Reference to the Login object
    Private m_oLogin As Kofax.Capture.DBLite.Login
```

```

'*** Reference to the RuntimeSession object
Private WithEvents m_oRuntimeSession As Kofax.Capture.SDK.CustomModule.IRuntimeSession

'*** Reference to the Active Batch
Private m_oActiveBatch As Kofax.Capture.SDK.CustomModule.IBatch

'*** Create a mutex to protect calls to ProcessBatch
Private m_oMutex As Threading.Mutex
'*** We will fire this event when we want to log something to the window
Public Event StatusMessage(ByVal Message As String)

Public Sub Initialize()
    Try
        '*** Initialize the Mutex
        m_oMutex = New Threading.Mutex

        '*** Create the Login object
        m_oLogin = New Kofax.Capture.DBLite.Login
        '*** Log into Kofax Capture
        m_oLogin.Login()
        '*** Set the App name and version
        m_oLogin.ApplicationName = "DeleteEventPage CM"
        m_oLogin.Version = "1.0"
        '*** Validate the User
        m_oLogin.ValidateUser("DeleteEvenPage.exe", True)
        RaiseEvent StatusMessage("Logged into Kofax Capture")
        '*** Cache the Runtime Session
        m_oRuntimeSession = m_oLogin.RuntimeSession
        '*** Add a handler function to handle Batch available event
        AddHandler m_oRuntimeSession.BatchAvailable, AddressOf RuntimeSession_BatchAvailable

    Catch ex As Exception
        RaiseEvent StatusMessage("An error occurred: " & ex.Message)
    End Try
End Sub
End Class

```

PageMarkedForDeletion Function

The `PageMarkedForDeletion` function returns a flag that indicates that the scan operator selected the page to be deleted. This flag is stored in a page-level custom storage string file.

```

Private Function PageMarkedForDeletion(ByRef oPageElem As
Kofax.Capture.SDK.Data.IACDataElement) As Boolean
    '*** Get the Page Custom Storage Strings
    Dim oPageCSSs As Kofax.Capture.SDK.Data.IACDataElement
    oPageCSSs = oPageElem.FindChildElementByName("PageCustomStorageStrings")
    '*** Get the Page Custom Storage String by element
    Dim oPageDeleteCSS As Kofax.Capture.SDK.Data.IACDataElement
    oPageDeleteCSS = oPageCSSs.FindChildElementByAttribute("PageCustomStorageString",
    "Name", "DeletePage")
    '*** Make sure we have a reference
    If Not oPageDeleteCSS Is Nothing Then
        '*** Check the value
        If oPageDeleteCSS.AttributeValue("Value") = "True" Then
            Return True
        End If
    End If
Return False
End Function

```

RuntimeSession_BatchAvailable Subroutine

This function handles the BatchAvailable event from Tungsten Capture. This handler attempts to process as many batches as possible. A mutex is implemented to ensure that this logic is processed by only one thread at a time.

```
Private Sub RuntimeSession_BatchAvailable()
    '*** Lock access to this call
    m_oMutex.WaitOne()
    Try
        '*** Process a new Batch
        While ProcessNewBatch()
            RaiseEvent StatusMessage("Process Batch Completed")
        End While
    Catch ex As Exception
        RaiseEvent StatusMessage("An error occurred: " & ex.Message)
    Finally
        '*** Release the mutex
        m_oMutex.ReleaseMutex()
    End Try
End Sub
```

ProcessNewBatch Function

The ProcessNewBatch function attempts to open the next available batch for this module. If it is able to open a batch, the batch is processed by removing the pages that the scan operator marked for deletion.

```
'*** This function returns True if we successfully process a Batch.
'*** Otherwise, we will return False (i.e. No Batches Available)
Private Function ProcessNewBatch() As Boolean
    Dim bResult As Boolean = False
    Try
        '*** Get the new Batch
        m_oActiveBatch = m_oRuntimeSession.NextBatchGet(m_oLogin.ProcessID, _
            KfxDbFilter.KfxDbFilterOnProcess Or KfxDbFilter.KfxDbFilterOnStates Or _
            KfxDbFilter.KfxDbSortOnPriorityDescending, _
            KfxDbState.KfxDbBatchReady Or KfxDbState.KfxDbBatchSuspended)
        '*** Make sure we have a reference
        If Not m_oActiveBatch Is Nothing Then
            RaiseEvent StatusMessage("Opened Batch: " & m_oActiveBatch.Name)
            '*** Extract the RuntimeDataElement
            Dim oRuntimeDataElement As Kofax.Capture.SDK.Data.IACDataElement = Nothing
            oRuntimeDataElement = m_oActiveBatch.ExtractRuntimeACDataElement(0)
            '*** Get the Batch element
            Dim oBatchElem As Kofax.Capture.SDK.Data.IACDataElement = Nothing
            oBatchElem = oRuntimeDataElement.FindChildElementByName("Batch")
            '*** Get the Pages element
            Dim oPagesElem As Kofax.Capture.SDK.Data.IACDataElement = Nothing
            oPagesElem = oBatchElem.FindChildElementByName("Pages")
            '*** Keep track of the Page Count
            Dim lngCount As Long = 0
            '*** Iterate through the Page elements
            Dim oPageElem As Kofax.Capture.SDK.Data.IACDataElement
            For Each oPageElem In oPagesElem.FindChildElementsByName("Page")
                '*** Increment our counter
                lngCount = lngCount + 1
                '*** Check if we should delete this page
                If PageMarkedForDeletion(oPageElem) Then
                    '*** Delete it!
```

```
oPageElem.Delete()
RaiseEvent StatusMessage("Deleted Page " & lngCount)
End If
Next
'*** Close the Batch
m_oActiveBatch.BatchClose(KfxDbState.KfxDbBatchReady, KfxDbQueue.KfxDbQueueNext, 0,
"")
'*** Return success
bResult = True
Else
RaiseEvent StatusMessage("No batches for me available.")
'*** Return false so we fall back into polling/waiting for signal
bResult = False
End If
Catch ex As Exception
RaiseEvent StatusMessage("An error occurred: " & ex.Message)
'*** If we have a reference to the Batch object, close in error
If Not m_oActiveBatch Is Nothing Then
m_oActiveBatch.BatchClose(KfxDbState.KfxDbBatchError, _
KfxDbQueue.KfxDbQueueException, 1000, ex.Message)
End If
End Try

Return bResult
End Function
```

Chapter 9

Creating an Export Connector

Export is the process of exporting images and data to long-term storage after all Tungsten Capture processing is finished. An *export connector* consists of two COM components that configure and execute this process. These two components could be in one or two .dll or .exe files.

Tungsten Capture includes the following standard export connectors:

- A database export connector that exports document index data to a Microsoft Access 97 or later, or an ODBC-compliant database. Source code for this export connector is installed to <Tungsten Capture installation folder>\Source\Export Connectors\Database.
- A text export connector that exports document index data to an ASCII text file. Source code for this export connector is installed to <Tungsten Capture installation folder>\Source\Export Connectors\KCEC-Text.

Both of these connectors export images, full text OCR files, and PDF files to the standard file system.

These export connectors are provided in Microsoft Visual Basic .NET. The source code for these export connectors is installed with Tungsten Capture. If you need to export document index data or files to other sources, you can modify one of the supplied connectors, create an entirely new one, or purchase one from Tungsten Automation or other third-party entities.

A wide variety of export connectors are available from Tungsten Automation for use with applications developed by Documentum, IBM FileNet, SharePoint, and others. Contact your Certified Solutions Provider for details on availability.

Export connectors are typically written in Visual Basic .NET, but they can also be written with any tool that supports the development of COM servers.

Tungsten Capture Export Type Library

The export connector API library is documented in the *Tungsten Capture Export Type Library API Reference*.

1. On your Tungsten Capture 10.2 installation media, navigate to the following location:

Documentation\Help\APIRef

2. Double-click **APIRefExport.chm**.

The *Tungsten Capture Export Type Library API Reference* appears in a browser window.

 If the *API Reference* does not display properly, copy APIRefExport.chm to a local drive.

Tungsten Capture and the Export Process

The Tungsten Capture Administration module manages the Export Setup process. It loads the export connector setup when the system administrator selects an export connector for a document class/batch class pair. The export connector setup must implement the KfxReleaseSetupScript COM interface required by the Administration module.

Similarly, the Tungsten Capture Export module manages the export process. It loads the appropriate export connector when the batch enters the Export queue. The export connector must implement the KfxReleaseScript COM interface required by the Export module.

The Database and the Text export connectors include the KfxReleaseSetupScript and KfxReleaseScript COM components in the ActiveX DLL.

When writing an export connector, you must provide two basic functions:

- Export connector setup, which is the user interface for configuring the export process within the Administration module. The export connector setup is called by the Administration module when the user selects an export connector to configure.
- Export, which performs the actual export of images and data to long-term storage. The export portion of the export connector is called by the Export module when it is exporting batches.

Requirements for the Export Connector Setup

The export connector setup component must define the KfxReleaseSetupScript COM interface.

The KfxReleaseSetupScript interface must include the following:

- One public object variable declared as ReleaseSetupData. The Administration module uses this variable to expose and update export configuration for the document class.
- Four methods called OpenScript, RunUI, ActionEvent, and CloseScript, which the Administration program calls to perform the various stages of the export setup process.

When you use the Administration program to select and configure an export connector the first time, it loads the export connector setup and does the following:

1. Fills in the document class properties in the ReleaseSetupData variable. These properties include the available document class index fields, batch class fields, and image file formats.
2. Calls the OpenScript method. You should use this method to perform any initialization required for the connector.
3. Calls the RunUI method. You should use this method to load a form or window that allows the user to configure the export process. For example, the database export connector displays a form that allows setup of database links, image formats, file folders, and so forth. When the user is finished with setup, you store the user's choices in the ReleaseSetupData variable and call the ReleaseSetupData.Apply method to save them permanently to the Administration database.
4. Calls the CloseScript method after the RunUI method is completed and the connector is about to be unloaded. You should use this method to perform any cleanup required for the connector.

If you make a change to the document class after the initial export setup process, the Administration program performs the same series of steps as before, with one exception: instead of calling the RunUI method, it calls the ActionEvent method with a set of parameters that indicate why it is being called. You should use this method to determine whether the change in the document class requires a change in the export setup process. For example, if a new index field is added, you might want to call the RunUI method to provide an opportunity to save this new data in the external database.

Requirements for the Export Connector

The Export module uses a COM interface called KfxReleaseScript to communicate with the export connector. The standard Text and Database export connectors define this interface in the Release.cls code module.

The KfxReleaseScript interface must include the following:

- One public object variable declared as ReleaseData. The Export module uses this variable to expose export data for the export process.
- Three methods called OpenScript, ReleaseDoc, and CloseScript, which the Export module calls to perform the various stages of the export process.

When a batch containing documents of a given class enters the Export module, it loads the export connector and does the following:

1. Fills in the general batch class and document class properties in the ReleaseData variable.
2. Calls the OpenScript method. You should use this method to perform any initialization required for the connector. The detail of the error occurring in the Export Connector, if any, is displayed in the batch history and the Tungsten Capture error log.
3. Fills in the properties specific to the first document to be exported in the ReleaseData variable and calls the ReleaseDoc method for the document. You should use this method to save the document data in the external database and copy the image files and full text OCR files to the selected export folders.
4. Repeats the process described in Step 3 for each remaining document to be exported.
5. Calls the CloseScript method after the last ReleaseDoc method is completed and the connector is about to be unloaded. You should use this method to perform any cleanup required for the connector. The detail of the error occurring in the Export Connector, if any, is displayed in the batch history and the Tungsten Capture error log.

Note that you cannot provide your own user interface for the export process. The Export module has its own user interface. The various methods listed earlier provide data to update the Export module user interface as documents and batches complete the export process. The export process is designed to run unattended.

ReleaseSetupData and ReleaseData Objects

The database export connector and text export connector support a wide variety of features, and both contain a long list of functions and subroutines. These functions are documented in the *Tungsten Capture Export Type Library API Reference*.

Writing an export connector is not actually as complex as you may think. Most of the required work is confined to specific locations in the connector. The code involves use of the methods and properties for the `ReleaseSetupData` and `ReleaseData` objects. Understanding these two objects is key to writing an export connector.

Export Connector Setup

There are only two places in the export connector setup where you have to write a substantial amount of code:

- The `RunUI` event should load a form that presents a user interface. This form can be as simple or as complex as you wish.
- When the user clicks OK to finish export connector setup, you must save the user's settings in the Tungsten Capture database. To do this, you must set up a links collection that specifies which index fields should be exported and then copy other settings as necessary into the `ReleaseSetupData` object. When this is finished, call the `Apply` method to save your changes.

ReleaseSetupData Object

The `ReleaseSetupData` object is a top-level object used by both the Tungsten Capture Administration module and the export connector setup to define the export process for a document class.

Some properties of this object are set up by the Administration module when the `ReleaseSetupData` object is created. For example, the available batch fields, index fields, image types, and storage types are set when the connector is loaded. The properties set up by the Administration module are read-only.

The connector must set the properties that identify the external data repository and the target export folders. It must also set the properties that establish the links between the available data fields and the fields or columns in the external data repository that will receive the document data during the subsequent export process. The properties will be available to the export connector.

Refer to the *Tungsten Capture Export Type Library API Reference* for details on the specific properties and methods available in the `ReleaseSetupData` object.

Export Connector

There is only one place in the export connector where you need to write a significant amount of code. The `ReleaseDoc` method is called every time the Export module is ready to export a new document. To export a document, you need to add code to this method that calls the following methods:

1. `ReleaseData.ImageFiles.Copy` copies the images to a location specified by the `ReleaseData` object.
2. `ReleaseData.TextFiles.Copy` copies the full text OCR files (if any) to a location specified by the `ReleaseData` object.

The index data must be exported last. The `ReleaseData.Values` collection contains values for all the index fields specified in the links collection during export connector setup, but there is no method provided for exporting index data, since the process varies widely from connector to connector. You are responsible for writing the proper code to export your index data to the appropriate repository.

3. `ReleaseData.CopyKofaxPDFFile` copies the PDF file that belongs to a document into the export PDF path that is defined during export connector setup.

`ReleaseData.CopyKofaxPDFFile` copies the PDF file that belongs to a document into the export PDF path that is defined during export connector setup.

ReleaseData Object

The `ReleaseData` object is a top-level object used by both the Tungsten Capture Export module and the export connector to access the batch and document class information and perform the actual document export.

All properties of this object are set up by the Export module. The properties that are common to all documents in the batch are set up when the `ReleaseData` object is initialized. Properties specific to an individual document are set up before the Export module calls the `ReleaseDoc` method for that document.

The connector must use the `ReleaseDoc` method to read the properties and save the index data and related document information in the external data repository. Then, the connector must copy the image files, any Tungsten Automation PDF files, and any full text OCR files to the target file system.

Refer to the *Tungsten Capture Export Type Library API Reference* for details on the specific properties and methods available in the `ReleaseData` object.

COM Servers: In-proc or Out-of-proc?

Export connectors can be developed using any language and tool that can create COM components. If you are working with a 32-bit development environment, COM components for both export connector setup and export can be designed as in-process servers (ActiveX DLLs) or as out-of-process servers (ActiveX EXEs).

Registering Your Export Connector

After completing your export connector, you must register it. Registration is a two-part process.

The `.inf` file, which should be located in the same folder as your compiled export connector, has a format similar to that of an `.ini` file. A sample `.inf` file is shown here.

```
[Scripts]
Custom Script
[Custom Script]
SetupModule=Custom.dll
SetupProgID=Custom.kfxreleasesetupscript
SetupVersion=1.0
ReleaseModule=Custom.dll
ReleaseProgID=Custom.kfxreleasescript
ReleaseVersion=1.0
SupportsNonImageFiles=True
RemainLoaded=True
SupportsKofaxPDF=True
SupportsOriginalFileName=True
SupportsMultipleInstances=False
```

The .inf file must contain a [Scripts] section that includes the name (up to 255 characters) of each connector you are registering. For each entry in the [Scripts] section, you must have a corresponding section with the entries listed in the table.

Inf File Entries

Inf File Entry	Description
ReleaseModule	Name of the compiled .exe or .dll containing the export COM component.
ReleaseProgID	Name of the export connector COM component. You can explicitly set the ProgID of your object using the ProgID Attribute. A ProgID is a unique identifier for every control, constructed through the combination of the control's project and object name (ProjectName.ObjectName).
ReleaseVersion	Version number assigned to the export connector COM component.
RemainLoaded	If True, the export connector remains loaded until processing is complete.
SetupModule	Name of the compiled .exe or .dll containing the export connector setup COM component.
SetupProgID	Name of the export connector setup COM component. You can explicitly set the ProgID of your object using the ProgID Attribute. A ProgID is a unique identifier for every control, constructed through the combination of the control's project and object name (ProjectName.ObjectName).
SetupVersion	Version number assigned to the export connector setup COM component.
SupportsKofaxPDF	If True, the export connector supports the output of Tungsten PDF files.
SupportsNonImageFiles	If True, the export connector supports eDocuments (non-image files).
SupportsOriginalFileName	If True, the export connector supports the use of the original name of the file.
SupportsMultipleInstances	If True, the export connector supports multiple instances of the Export service. Note that the Multiple Instance Support feature is only available for Tungsten Capture Enterprise users.

1. Create an .inf file for your export connector.
2. In the Administration module, on the **Tools** tab, in the **System** group, click **Export Connectors**.
3. On the **Export Connector Manager** window, select the connector to register.
4. Click **Add**, browse to your .inf file, and click **Open**.
5. Select the connector to use and click **Install**.
6. Click **OK**, and your export connector is registered.
7. Close all windows to complete the process.

Scripting in a High Availability Environment

When writing connectors for Tungsten Capture in a high availability environment, you should observe the error handling guidelines found in [High Availability Environments](#).

For more information on high availability, refer to the *Tungsten Capture Installation Guide*.

Chapter 10

Deploying Customizations

As a developer, you can use the Tungsten Capture Deployment Service to easily and automatically deploy customizations throughout the entire Tungsten Capture installation. Tungsten Capture customizations include:

- Workflow agents
- Custom panels
- Custom modules
- Export connectors

The Tungsten Capture Deployment Service deploys Tungsten Capture customizations to any client workstation that is connected to a server with the service installed and enabled.

You can develop and test a customization on one node of the Tungsten Capture system and then enable the deployment and registration to other client workstations and remote sites.

Once you configure a customization update on a Tungsten Capture server, it can be automatically downloaded and installed to client workstations connected to that server. Also, by deploying a customization update to a Tungsten Capture Network Server central site, the RSA agent automatically updates remote servers from a single source on the central site.

Deploying customization updates does not force operators to stop work. If an operator is using Tungsten Automation applications that require a customization update, a message displays indicating that updates are available. Operators can decide when to reboot and automatically trigger the update.

Also, workstations installed after an initial deployment can obtain posted customizations. The service provides a comprehensive log of all deployment activity.

The Tungsten Capture Deployment Service does not:

- Automatically delete customizations from workstations.
- Automatically deploy customizations before a batch needs it; deployments are set up by central site administrators.
- Deploy other types of Tungsten Automation software such as export connectors or Tungsten Transformation Modules.
- Deploy other types of generic software; it deploys only Tungsten Capture customized software.

Installing the Customization Deployment Service

For any workstation that requires customization deployment, you must manually install the provided Administration service named "KCDeploymentService.exe." Run this service as a user with read-write access to the `Customization Deployment` folder on the server and with administrator rights to allow files to be copied and replaced in `<Administration installation folder>\Bin`.

The service installation starts a background program that, if necessary, alerts users to restart their computer so that a customization deployment can finish. The background program displays an icon in the Windows notification area.

1. At a command prompt, run the Microsoft installation utility (InstallUtil.exe) with the Tungsten Automation Deployment Service as a parameter. If you installed Administration in the default location, type the following and press Enter:

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe "C:\Program Files (x86)\Tungsten Automation\Capture\Bin\KCDeploymentService.exe"
```

The **Enable Tungsten Capture Deployment Service** window appears.

2. Click **OK**.

Command Line Parameters

The Tungsten Capture Deployment Service supports the following command line parameters.

 When using command line parameters, the name of the service (KCDeploymentService.exe) must be the last parameter on the command line.

/username="[user name]"

You can specify the Windows account to be used to run the service with the username parameter, along with the password, as shown in the following example:

```
InstallUtil.exe /username="<MyDomain>\<MyName>" /password=xj987L2  
KCDeploymentService.exe
```

/password=[password]

Use this parameter to specify the password for the user account.

/unattended

Use this parameter to suppress the Enable Tungsten Capture Deployment Service window.

/u or /uninstall

Use this parameter to remove the Tungsten Capture Deployment Service.

Setting Up a Customization Deployment

Setting up a customization deployment involves creating subfolders in a server's `CaptureSV` folder, placing your customization files in appropriate subfolders, and creating a time file, as explained in the following procedure.

1. On a server, create the following folder:

```
\\<server name>\CaptureSV\Customization Deployment
```

2. In the `Customization Deployment` folder, create a folder for the customization files to deploy. For example, you might create a folder named `MyCustomModule`.

This folder should contain the executables and dlls to deploy to multiple workstations. The following shows how one folder might contain both `Custom.exe` and `Custom.dll`.

```
\\<server name>\CaptureSV\Customization Deployment\<MyModule>\Custom.exe
```

```
\\<server name>\CaptureSV\Customization Deployment\<MyModule>\Custom.dll
```

You can create more than one folder for deployment. For example, you might create three folders containing different customization files.

```
\\<server name>\CaptureSV\Customization Deployment\<MyCustomModule_1>\
```

```
\\<server name>\CaptureSV\Customization Deployment\<MyCustomModule_2>\
```

```
\\<server name>\CaptureSV\Customization Deployment\<MyWorkflowAgent_1>\
```

3. If any of your customization files require registration, you must create a batch file named "Registration.cmd" in your customization folder. The batch file contains any scripting needed to register your files.

Registration may include both Windows components and Tungsten Automation files.

Therefore, if Microsoft registration utilities (`Regasm.exe` or `Regsvr32.exe`) or the Tungsten Capture Extension Registration Utility (`RegAscEx.exe`) are required, they must be available on the target system. Or, you may want to include them in the deployment so that availability and system paths are not factors.

```
call regsvr32.exe /s "C:\Program Files (x86)\Tungsten Automation\Capture\Bin
\SampleWorkflowOcx.ocx"
if errorlevel 1 exit errorlevel
call regsvr32.exe /s "C:\Program Files (x86)\Tungsten Automation\Capture\Bin
\WFAgent.dll"
if errorlevel 1 exit errorlevel
call RegAscEx.exe /f "C:\Program Files (x86)\Tungsten Automation\Capture\Bin
\WorkflowAgentWithOCX.AEX"
if errorlevel 1009 exit errorlevel
if errorlevel 1008 exit 0
```

You should write the script so that a successful exit returns a zero. Any exit failure should return a non-zero value.

During the deployment process, a registration script has 25 seconds to finish. If it does not finish, the deployment process fails. The deployment service does not terminate any registration script process, so if a registration script unintentionally creates an endless loop, you must manually stop it.

4. Create a file named "time.txt" in the `Customization Deployment` folder. The first line must contain a string that specifies the local time to deploy the customizations. String format:

```
hh:mm
```

The "hh" is the number of complete hours that have passed since midnight, and "mm" is the number of complete minutes since the start of the hour. The "hh" must be a value from 00 to 23, and "mm" must be a value from 00 to 59. If the time file does not follow the correct format, no deployments occur.

The specified time is an approximate time for deployment. The deployment occurs the first time the deployment service checks the folders after the specified time.

This time applies to all customizations across all sites. Each deployment service running at each workstation deploys the customizations at the specified local time.

If the time.txt file does not exist, no deployments occur.

Initiating a Customization Deployment

When your customization files are in place, you are ready to initiate the deployment. Initiating the deployment involves creating a version file (Version.txt) and placing it in the customization folder, as explained below.

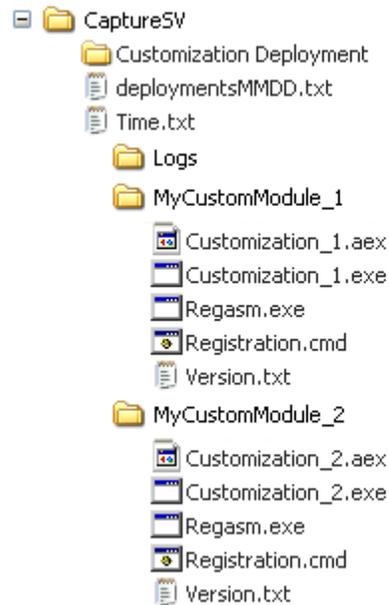
1. When you are ready for workstations to deploy a customization, create a file named "Version.txt" and enter a version indicator on the first line.

The Administration Deployment Service uses the first line of this file to uniquely identify the customization during status logging. Any additional lines are ignored. Use the version string to specify a version of the customization files so that you can distinguish one update from another. The version string can be empty or lengthy. Its length is not checked or truncated.

When you create or modify this file, it signals to the deployment service that all files are in place and ready to be deployed. The Administration Deployment Service uses the time stamp of this file to coordinate deployments and prevent customizations from being repeatedly deployed.

2. Perform one of the following:
 - For a first time deployment, place a separate Version.txt file in each of your customization folders.
 - For subsequent deployments, open Version.txt, update the version string, and save the file. This gives the file a new time stamp and signals to the service that an updated deployment is ready.

The following figure shows the contents of a sample Customization Deployment folder.



Sample Customization Deployment Folder

Viewing Customization Deployment Status

When a customization is deployed to a workstation, the deployment service adds a status entry to a log file named "deployedMMYY.txt" in the `Customization Deployment\Logs` folder on the server. MMYY is the two-digit month and year. The service automatically creates a new file at the beginning of each month.

Each entry in the file is a single line with the following comma-separated values:

```
<Station ID>,<deployment name>,<deployment version string>,<status>
```

Use the comma-separated values to import the file into Excel or another spreadsheet application for sorting or other analysis. The Status Values table describes each value.

i If the `\\<server name>\CaptureSV\Customization Deployment\Logs` folder is inaccessible, the deployment service writes to a local log file at `<Application Data>\Tungsten Automation\Capture\Local\Customization Deployment\Logs`.

Status Values

Value	Description
Station ID	Station ID of the workstation where the deployment was attempted.
Deployment name	Customization deployment folder name.

Value	Description
Deployment version string	First line from the version.txt file in the customization deployment folder.
Status	Completed <date/time> Pending reboot <date/time> Failed <date/time>: <error> Where <error> is the error string returned when the file deployment fails.

Deploying Customizations While Applications Are Running

When the deployment service detects that customization files on a workstation are in use, it attempts to determine if they are locked only by service processes, or if they are locked by some combination of services and interactive applications such as Administration or Validation.

If customization files are only locked by services, the deployment service will temporarily stop the services, deploy the customization, and restart the stopped services once the deployment is complete.

If the deployment service detects that customization files are locked by interactive applications, it does not stop any processes. Instead, the completion of the deployment and the final move of the locked files take place on the next system restart.

In this case, the following actions occur on the workstation where the restart is required:

The deployment service:

- Sets any service that is currently locking customization files to manual startup but leaves it running.
- Adds an entry to the deployment log file on the server with status indicating that deployment will complete pending a reboot.
Schedules the customization files to be moved on the next reboot (when the deployment service starts up).

The customization deployment background program in the Windows notification area:

- Displays a message to the user indicating that updates are ready and a system reboot is necessary. The notification message displays once every 24 hours.

The user can:

- Ignore the update message. If the user ignores the notification, the background program continues to run and displays the message 24 hours later.
- Close the background program. If the user closes the background program, it no longer displays an update message, but it runs again at system startup.
- Reboot the system. If the user reboots the system, the updates take place.

i When a user reboots a workstation, the deployment service only deploys customizations that were pending a reboot or that failed on the last deployment attempt.

About the Customization Deployment Process

This section explains the basics of the customization deployment process for system administrators.

Administrator Actions

The following actions are performed by a system administrator on the server.

- To prepare for a deployment, an administrator creates and populates the required files in `\<server name>\CaptureSV\Customization Deployment`.
- To initiate a deployment, an administrator creates or updates one or more `Version.txt` files.

Deployment Service Actions

The following actions are performed by the Tungsten Capture Deployment Service.

- On the server, the Tungsten Capture Deployment Service copies each customization folder that contains a version file in `\\<server name>\CaptureSV\Customization Deployment` to a corresponding folder on each workstation in `<Application Data>\Kofax\Capture\Local\Customization Deployment`.
- On each workstation, the Tungsten Capture Deployment Service copies customization files from `<Application Data>\Kofax\Capture\Local\Customization Deployment` to the workstation's Tungsten Capture Bin folder (`<Tungsten Capture installation folder>\Bin`) and executes the registration scripts.
- Locally, on each workstation, at various steps in the deployment process, the deployment service writes to a file called "DeploymentStatus.txt" in `<Application Data>\Tungsten Automation\Capture\Local\Customization Deployment` on the client station. This file contains a value that corresponds to one of the following status values.

Value	Description
0	Customization has not been deployed.
1	Customization was or is in the process of deployment.
2	Customization requires a system reboot to complete.
3	Customization has successfully been deployed.
4	Customization has failed to deploy.

i Do not modify the contents of `DeploymentStatus.txt`. The deployment service uses this file to determine at what stage the deployment is in the process.

- After the deployment, the Tungsten Capture Deployment Service reports status to a log file on the server named "deploymentsMMDD.txt" located in `<server name>\CaptureSV\Customization Deployment\Logs`.

Tungsten Capture Network

When synchronization occurs between the central site and remote sites, deployment files are copied to the Tungsten Capture Network folder at remote sites. Customization files then become accessible to remote workstations. A synchronization does not result in immediate deployments. The time that a customization is deployed is still determined by the time file.

Status

Each remote site's deployment log will update to the central site as needed. When status is returned from the remote sites to a central site, the following folder convention is used:

```
\\<server name>\CaptureSV\Customization Deployment\Logs\<Site name><Site GUID>\deployedMMYY.txt
```

Appendix A

Custom Module Sample

This appendix walks you through the process of installing, registering, and launching the sample custom module provided in your Tungsten Capture product. Tungsten Capture includes a module similar to Quality Control for use as a starting point for a custom module. By using this module as a starting point, the task of creating a custom module is greatly simplified. You can take advantage of the existing panel, menus, and other features of the standard module to create your own module. The standard module also includes support for scanning, which is normally a difficult feature to implement.

This generic sample module is intended to demonstrate some simple options for opening and closing a batch. The sample module does not perform any batch processing functions. Files for this feature (CustomStandard.exe, CustomStandard.aex, and Online Help IDs.doc) are located in <Tungsten Capture Installation folder>\Source\Sample Projects\CustMod\CustomStandard.

The folder contains the files necessary to register a custom module named "Sample." The sample gives a generic example of a custom module, which you can install and register for demonstration purposes.

The differences between using a custom standard module and using the generic custom module interface are listed in the table.

Difference Between Custom Standard Module and Custom Module

	Custom Standard Module	Custom Module
Starting Point	To create a custom standard module, first copy the Tungsten Capture CustomStandard.exe to a unique name. There is no need to create a Visual Basic .NET project and start from scratch.	A custom module is created as a new project and written in Visual Basic .NET.
Data Access	Tungsten Capture Module .NET Type Library (Kofax.Capture.CaptureModule.dll)	Tungsten Capture Document Access .NET Type Implementation Library (Kofax.DBLite.dll) Tungsten Capture Optimized Custom Module .NET Type Implementation Library (Kofax.DBLiteOpt.dll) Runtime XML - read/write Setup XML - read-only

	Custom Standard Module	Custom Module
User Interface	<p>Any Tungsten Capture interface elements in the custom standard module can be suppressed.</p> <p>The following UI elements are included.</p> <ul style="list-style-type: none"> • Image Viewer • Image Tools Toolbar • Batch Contents Panel • Batch Thumbnails Panel • Scan Controls Panel • Notes Panel • Batch Tools Toolbar • Batch Navigation Toolbar • Batch Navigation Toolbar • Batch Filters Toolbar <p>Customizations are done using OCXs.</p>	User interface is entirely written by the developer.
Advantages	<p>Starting with a Tungsten Capture module provides:</p> <ul style="list-style-type: none"> • Batch editing and scanning • An image viewer • Batch selection • Scan/import 	The module is designed and implemented by the developer.
Disadvantages	All added code must use the OCX interface.	The developer must write the entire application.
Registration in Tungsten Capture is the same for both.	RegAscEx.exe registers the .exe as a custom module.	RegAscEx.exe registers the .exe as a custom module.
Licensing	A Concurrent Station License is required only if pages are added.	A Concurrent Station license is required only if pages are added.

Licensing

Custom standard modules use the Tungsten Capture licensing mechanism; a Concurrent Station License is required.

Tungsten Capture Optimized Custom Module .NET Type Implementation Library

The Tungsten Capture Optimized Custom Module .NET Type Implementation Library is an easy-to-use custom module interface that supports fast retrieval and selective update of data. This library provides a mechanism for using the Tungsten Capture Document Access .NET Type Implementation

Library (Kofax.Capture.DBLite) to select and open a batch. However, once open, it is not necessary to export the data to XML.

Instead, the code can retrieve an API object that emulates an XML element, but without XML. The code can selectively extract and update Tungsten Capture data through this API. Because the data is processed selectively, execution is faster than XML import and export. The actual organization of Tungsten Capture data (the element and attribute structure) is identical between the XML and this new API.

The Tungsten Capture Optimized Custom Module .NET Type Implementation Library is intended for use with both custom modules and workflow agents. Because of the performance overhead with XML, we suggest that you use the optimized API whenever possible.

Tungsten Capture continues to support XML, which may still be better for developers with applications that are well-suited to an XML interface.

The Tungsten Capture Optimized Custom Module .NET Type Implementation Library is implemented in the Tungsten Capture Optimized Custom Module .NET Type Implementation Library (Kofax.DBLiteOpt.dll).

The complete API for the Tungsten Capture Optimized Custom Module .NET Type Implementation Library is documented in the *Tungsten Capture API Reference*, which is available from the `Documentation\Help\APIRef` folder on your Tungsten Capture installation media.

Tungsten Capture Document Access .NET Type Implementation Library

This section gives you information about Tungsten Capture Document Access .NET Type Implementation Library, the component that offers the ability to integrate a custom module into your Tungsten Capture installation. Tungsten Capture Document Access .NET Type Implementation Library makes it possible for the custom module to access batch information from Tungsten Capture. Likewise, your custom module can relay batch information to Tungsten Capture. The Tungsten Capture Document Access .NET Type Implementation Library, Kofax.DBLite.dll, is available from `<Tungsten Capture installation>\Bin`.

Tungsten Capture Document Access .NET Type Implementation Library gives you the ability to do the following:

- Route a Tungsten Capture batch to a custom module for processing.
- Use XML transport files to export/import batch information between Tungsten Capture and a custom module.
- Close a batch from a custom module and route it to the next module in the Tungsten Capture workflow.

Although it is possible to access and modify batch data using Tungsten Capture Document Access .NET Type Implementation Library and XML, we recommend that you use the Tungsten Capture Optimized Custom Module .NET Type Interface Library because it provides a more efficient mechanism to selectively retrieve and manipulate data. A degradation in performance could occur when using XML to import and export data.

The complete API for the Tungsten Capture Optimized Custom Module .NET Type Implementation Library is documented in the *Tungsten Capture API Reference*, which is available from the Documentation\Help\APIRef folder on your Tungsten Capture installation media.

 Some API methods and properties are reserved for exclusive use by Tungsten Automation. These APIs are not documented and their use is not supported. These items are not displayed when browsing the API definitions.

Development Environment

To develop a custom module that is compatible with Tungsten Capture, you need the following tools for the setup OCX (optional) and runtime executable.

Setup OCX

- Tool such as Visual Basic .NET that allows you to implement a COM interface. Custom module testing with Tungsten Capture has been certified with Visual Basic .NET.
- Tungsten Capture Administration module type library. Located in <Tungsten Capture installation folder>\Bin, the file is Kofax.Capture.AdminModule.dll.

Runtime Executable

- Tool such as Visual Basic .NET to create an instance of a COM interface. Custom module testing with Tungsten Capture has been certified with Visual Basic .NET.
- Tungsten Capture Document Access .NET Type Implementation Library, the COM object that allows your custom module to access batch information from Tungsten Capture, also allows your custom module to relay information about batches to Tungsten Capture. The Tungsten Capture Document Access .NET Type Implementation Library, Kofax.DBLite.dll, is available from <Tungsten Capture installation>\Bin.
- Tungsten Capture Optimized Custom Module .NET Type Implementation Library, the API that allows your custom module to selectively and quickly retrieve data from Tungsten Capture. This API provides the mechanism to use the Tungsten Capture Document Access .NET Type Implementation Library used to select and open a batch. The Tungsten Capture Optimized Custom Module .NET Type Implementation Library, Kofax.DBLiteOpt.dll, is available from <Tungsten Capture installation>\Bin.

Although it is possible to access and modify batch data using Tungsten Capture Document Access .NET Type Implementation Library and XML, we recommend that you use the Tungsten Capture Optimized Custom Module .NET Type Implementation Library, because it provides a more efficient mechanism to selectively retrieve and manipulate data. A degradation in performance could occur when using XML to import and export data.

Installing the Sample Custom Module

This section explains how to install the files required to register and run the sample custom module.

To customize Tungsten Capture, you must have Administrator privileges to install files to the Tungsten Capture installation folder.

1. Start Windows Explorer and browse to `<Tungsten Capture installation folder>\Source\Sample Projects\CustMod\Generic`.
2. Display the contents of the `Generic` folder.
3. Copy the following files to `<Tungsten Capture installation folder>\Bin`:
 - `CMSample.aex`
 - `CMSample.exe`

Registering the Sample Custom Module

This section explains how to use the Custom Module Manager to register the sample custom module, so you can use it as an Tungsten Capture processing queue.

 You can also register custom modules with the [Tungsten Capture Extension Registration Utility](#).

After completing the registration process, you can view the properties for the custom module sample. The properties are defined in the registration file (`CMSample.aex`), which you copied earlier to the `Bin` folder. For detailed information about the format for the registration file, see [Registration File Creation](#).

1. In the **Administration** module, on the **Tools** tab, in the **System** group, click **Custom Modules**. The **Custom Module Manager** window appears.
2. On the **Custom Module Manager** window, click **Add**. The **Open** window appears.
3. On the **Open** window, browse to the `Bin` folder, select **CMSample.aex**, and click **Open**. The **Custom Modules** window appears.
4. On the **Custom Modules** window, select **Sample** and click **Install**. A message appears to confirm that the registration process is complete.
5. Click **OK** to clear the confirmation message.
6. Click **Close** to exit the **Custom Modules** window. The name of the newly registered custom module "Sample" appears on the **Custom Module Manager** window.

Adding the Sample to the Batch Processing Workflow

Once a custom module is successfully registered, it is available for selection as a batch class processing queue from the Create Batch Class and Batch Class Properties windows.

You can select it as a processing queue for a batch class, just like any other processing queue. Just select the custom module from the list of Available Queues and click **Add**.

When you move the custom module to the list of Selected Queues, it is inserted according to the valid processing order, according to its processing function. The valid processing order is defined by the Follow and Precede entries that appear on the Custom Module Properties Advanced tab (see the preceding section).

For example, the Sample custom module is defined to follow Scan and precede Export. Therefore, you can move it to any position within the list of Selected Queues after Scan and before Export.

When you publish the batch class with the custom module in its workflow, check the results log on the Publish window. Be sure that no errors or warnings are associated with the custom module.

1. Use the Administration module to create or open the batch class to include the sample custom module in its workflow.
2. Select the batch class name on the **Definitions** panel and right-click to open the **Properties Queues** tab.
Notice that "Sample" appears on the list of **Available Queues**.
3. Verify that Scan appears on the **Selected Queues** list, and then add **Sample** to the list. Click **OK** to close the window.
4. Publish the batch class.

Using Batch Manager with a Custom Module

From Batch Manager, you can view the status of any batch, including a batch that includes a custom module in its workflow. You can launch the custom module and process the batch from Batch Manager.

In addition, the custom module is available from the list of queues for batches that have the custom module defined as part of their workflow. You can optionally redirect a batch to a custom module from Batch Manager.

Creating a Batch to Open in the Sample Custom Module

This section explains how to create a batch to be routed to the sample custom module.

1. Start the Scan module and create a batch based on the batch class you published.
2. Scan the batch and close it.
3. Do one of the following to start the sample module:
 - Double-click CMSample.exe in <Tungsten Capture installation folder>\Bin.
 - Start Batch Manager, select the batch that is ready to open in the sample custom module, and on the **Home** tab, in the **Batch** group, click **Process Batch**.

The **Sample** custom module starts and appears on the desktop. If you use Batch Manager to start the custom module, the options for opening a batch are slightly different.

4. Optionally select the **Include Setup Data** check box to import the batch class setup data, along with the batch data.

The setup data includes information about the batch class properties, including export connectors, image cleanup, recognition profiles, and more. For this demonstration, the setup

data includes the standard administrative data. Your implementation will probably include a setup OCX to provide batch class configuration settings and publish checks associated with a custom module.

5. Select **By Batch** as the process mode. For information about the "By Document" process mode, see [Processing by Document](#).
6. If you launched the sample from **Batch Manager**, click **Open Batch #** (where # is the incremental batch identification number) to open the batch.
If you did not launch the sample from **Batch Manager**, do one of the following:
 - Click **Open Next Batch** to open the next available batch.
 - Click **Select Batch** to display the **Open Batch** window where you can select a batch, and click **OK**.

Processing by Document

The previous section explained how to open a batch in the Sample custom module using the By Batch process mode. You can also process a batch with the By Document process mode. Use this mode to open a specific range of documents in a batch, rather than the entire batch.

1. In the Scan module, create a batch as described in [Creating a Batch to Open in the Sample Custom Module](#). Be sure that the batch includes documents.
2. Start the sample custom module as described in [Creating a Batch to Open in the Sample Custom Module](#).
3. When the sample module window appears, select **By Document** as the process mode and select the **Include Setup data** check box.
4. Select the option to open the batch.
5. At "Open documents," select a range of documents to process in the sample custom module. If the batch has no documents, the options for selecting and opening a range of documents are grayed on the window.
6. Click **Open Documents**.
7. Under **Process Documents**, click **Copy XML** or **Corrupt XML**. For more information, see [XML Transport Files](#).
8. Click **Close Documents**.
9. Under **Process Batch**, click **Copy XML** or **Corrupt XML**.
10. Click **Close Batch**.

Setting the Batch Custom Storage String

Use the fields in the Batch Custom Storage String area to read or write the batch custom storage string when a batch is opened. If the batch is closed, you can only read the value.

1. In Batch Manager, process a batch that uses the CMSample custom module.
2. Open the batch using the Sample window, using the "Open Batch *n*" button.
3. Type a name and a value in the appropriate fields.
4. Click **Set**.

The name and value are set in the batch.

Getting the Batch Custom Storage String

1. In Batch Manager, process a batch that uses the CMSample custom module.
2. Open the batch using the Sample window, using the "Open Batch *n*" button.
3. Type the name of the batch custom storage string in the name field.
4. Click **Get**.
The value appears.

XML Transport Files

XML transport files are used to relay batch class (setup/administration) and batch (runtime) information between Tungsten Capture and the custom module. For example, the sample custom module uses a runtime XML file called RTExport.xml to list Tungsten Capture database information that the custom module uses. When batch information is imported back to the database after processing in the custom module, another XML file called RTImport.xml is used. Any modifications that you make to the XML format are validated by Tungsten Capture to ensure compatibility.

The sample module uses the standard Tungsten Capture configuration settings. You will probably implement a setup OCX to define additional custom configuration and property settings associated with custom module functionality. The configuration information is relayed between Tungsten Capture and the custom module via files called SUExport.xml and SUImport.xml.

Once the batch is open, the following files are generated in `ProgramData\Tungsten Automation\Capture\Local\Kofax.Sample`.

- **RtExport.xml**: Contains Tungsten Capture database information related to the current batch. When the "By Batch" process mode is enabled, RtExport.xml includes data related to all the documents in the batch. When the "By Document" process mode is enabled, a separate file (DcExport.xml) is generated with the document information.
- **SuExport.xml**: Contains configuration settings and publish checks associated with the custom module. This file is generated if you selected the "Include Setup data" check box before opening the batch.
- **DcExport.xml**: Contains Tungsten Capture database information related to selected documents in the batch. This file is generated only if you select the "By Document" process mode.

A set of Document Type Definition (DTD) files is also generated. The DTD files list the required XML format that Tungsten Capture requires: AcBatch.dtd, AcDocs.dtd, and AcSetup.dtd. (AcDocs.dtd is generated only if you opened the batch in "By Document" mode.)

The .dtd files appear in the Tungsten Capture installation folder. To examine these DTD files in detail, open them in any text editor.

Copying the XML Files Back to Tungsten Capture

1. Click the **Copy XML** button.

Notice that additional import files are generated in the Kofax.Sample folder: DcImport.xml, RtImport.xml, and SuImport.xml. These files are used to import batch information to Tungsten Capture from the sample module.

2. Optionally, select a state (Ready, Suspended, Error) in which the batch should be closed.
3. Click the **Close Batch** button. The batch is routed to the next module in the Tungsten Capture workflow, unless it is suspended or closed in error.

Corrupt XML

Even if the XML format is technically valid and well-formed, the content may not be compatible with Tungsten Capture. The Sample custom module includes a mechanism to demonstrate the effect of XML content that is incompatible with Tungsten Capture. You can click the Corrupt XML button to intentionally insert an incompatible text string in one of the XML import files.

If you are processing in the By Document mode, you can click the **Corrupt XML** button to insert an incompatible text string in DcImport.xml. If you are processing in By Batch mode, click the **Corrupt XML** button to insert a similar string in the RtImport.xml file.

When you select **Corrupt XML**, the batch is suspended when you close it. A message appears with text explaining that the error is related to the import process, the KofaxCaptureRuntime root element, and the first batch element.

When the XML format is not compatible with Tungsten Capture, the batch is not routed to the next queue in the Tungsten Capture processing workflow.

Create Page

Use the Create Page button to create a page or a new document. To create a page or a document, the Process mode must be set to "by Batch."

Clicking the Create Page button adds a new page to the XML file. If "Create page in new document" is selected, a document and page are added to the XML file.

You can open RtImport.xml in the Kofax.Sample folder to see the added lines.

Appendix B

Workflow Agent Sample

This appendix walks you through the process of installing, registering, and using the sample workflow agent provided in your Tungsten Capture product package. The name of this sample is "Validation Workflow Agent."

A workflow agent is a custom application that has the ability to examine and modify batch data, as well as change the routing (next module) and status for the batch. Workflow agents are invoked whenever a batch is closed from any module.

Workflow agents consist of a runtime module and optional setup program. Prior to using a workflow agent, you must write the necessary code, and then register the agent.

You can find the sample workflow agent application in `\Kofax\Capture\Source\Sample Projects\Workflow\WFSample`. This folder contains a Visual Basic .NET Project for both a Workflow Agent and Setup OCX. This folder also contains a compiled version of this Project, and the associated sample Tungsten Capture Registration File (.aex).

The sample supports a setup OCX that adds a new menu item (Validation Workflow Properties) to the Batch Class context menu.

At runtime, the Validation Workflow Agent skips validation for a document if all index fields in that document are automatically recognized with at least a user-specified confidence level. You can specify the confidence level with extensions added by the Workflow Agent setup OCX. If all documents in a batch meet the confidence level, the Validation module (and Verification module, if applicable) is skipped.

 Workflow agents are very similar to custom modules, and the windows for registering workflow agents are nearly identical to those for custom modules.

Installing the Sample Workflow Agent

This section explains how to install the files required to register and run the sample workflow agent.

If you are customizing Tungsten Capture, you must have Administrator privileges to install files to the Tungsten Capture installation folder.

1. Start Windows Explorer and browse to `Kofax\Capture\Source\Sample Projects\Workflow\WFSample`.
2. Display the contents of the `WFSample` folder.

3. Copy the following files from the `WFSample` folder to `<Tungsten Capture installation folder>\Bin`:
 - `WorkflowAgentSample.net.aex`
 - `SampleWorkflowSetup.net.dll`
 - `WFAgent.net.dll`

Registering the Sample Workflow Agent

This section explains how to use the Workflow Agent Manager to register the sample workflow agent, so you can use it in Tungsten Capture.

 You can also register workflow agents with the Tungsten Capture Extension Registration Utility.

1. In the **Administration** module, on the **Tools** tab, in the **System** group, click **Workflow Agents**. The **Workflow Agent Manager** window appears.
2. On the **Workflow Agent Manager** window, click **Add**. The **Open** window appears.
3. On the **Open** window, browse to the `Bin` folder, select `WorkflowAgentSample.net.aex`, and click **Open**. The **Workflow Agents** window appears.
4. On the **Workflow Agents** window, select **Validation Workflow Agent** and click **Install**. A message appears to confirm that the registration process is complete.
5. Click **OK** to clear the confirmation message.
6. Click **Close** to exit the **Workflow Agents** window. The name of the newly registered **Validation Workflow Agent** appears in the **Workflow Agent Manager** window. You can now view the properties for the registered workflow agent. The properties are defined in the registration file (`WorkflowAgentSample.net.aex`), which you copied earlier to `<Tungsten Capture installation folder>\Bin`.
7. Click **Properties** to open the **Workflow Agents Properties** window - **General** tab.
8. Click the **General** and **Programs** tabs to familiarize yourself with the other properties for the sample workflow agent, and then click **Close**.
9. Click **Close** again from the **Workflow Agent Manager** window.

Using the Sample Workflow Agent

This section explains how to use the sample workflow agent in a batch class.

1. In the Administration module, create or open a batch class for which you want to include the sample workflow agent.
2. Select the batch class name from the **Definitions** panel and right-click to open the **Batch Class Properties** window - **Queues** tab.

3. Ensure that Scan, Validation, and other modules appear as applicable on the **Selected Queues** list.
4. On the **Batch Class Properties** window - **Workflow Agents** tab, select **Validation Workflow Agent** and click **Add** to move it to the **Selected Workflow Agents** list.
5. Click **OK** to close the **Batch Class Properties** window.
6. On the **Definitions** panel, right-click to display the context menu, and then click **Validation Workflow Properties**.
The **Validation Workflow Properties** window appears.
7. Select **Skip validation if the confidence for all fields is at least**.
8. Move the slider to the desired confidence level. If the confidence level for all the documents in your batch meets or exceeds this setting, the Validation module (and Verification module if it is part of the batch class) is skipped.
9. Click **OK**.
10. Publish the batch class.

Appendix C

Converting Script Code from SBL to VB.NET

Upgrading from SBL to VB.NET in Tungsten Capture offers several advantages, including more powerful scripting capabilities, enhanced error handling, and access to the .NET framework for integration and data processing. While the upgrade requires rewriting scripts, the benefits in terms of performance, flexibility, and maintainability often make it worthwhile for complex workflows. The process involves understanding the existing SBL logic, rewriting it in VB.NET, and testing each step to ensure that the new code functions correctly in Tungsten Capture.

In the conversion tables and examples that follow, the "Me" object refers to the current instance of the script class being executed. This object can be derived from Document Validation, Folder Validation, Field or Recognition Script, depending on the context in which the script is running.

Variables

SBL variables or variable attributes	VB.NET
KfxBatchName	Accessed through argument parameters (e.g: e As PostDocumentEventArgs, Or use: e As BatchEventArgs) of the "Me" object's event handlers (e.g: DocumentPreProcessing event handler): e.Document.Batch.Name Or use: e.Batch.Name
The first parameter VerifyBatch (As Integer) of the function KfxLoadValidation	Me.Application.ToString (Values: "Verify" for VerifyBatch <> 0, or "Validation" for VerifyBatch = 0)
KfxBatchId	e.Batch.ExternalID
KfxBatchClassId	e.Batch.ClassID
KfxClassName	e.Document.ClassName
KfxDocClassId	e.Document.ClassID
KfxFolderClassName	e.Folder.ClassName
KfxFolderClassId	e.Folder.ClassID
KfxPageFile	e.Document.ActivePage.FileName for the recognized page file Or use: e.Document.ActivePage.OriginalFileName for the scanned page file

SBL variables or variable attributes	VB.NET
KfxAcmDocument.PageCount	e.Document.PageCount
KfxAcmDocument.Note	e.Document.Note
KfxErrorMessage	We can set an error message e.Document.Note by throwing a RejectAndSkipDocumentException
KfxField_Name	<IndexFieldVariableAttribute("Field Name")> _ Dim WithEvents Developer_Field_Name As FieldScript And access the value like that: Developer_Field_Name.IndexField.Value
KfxCLField_Name	Developer_Field_Name.IndexField.Confidence
Kfx_KFX_Field_Type_Name	Me.IndexField.Value
KfxCL_KFX_Field_Type_Name	Me.IndexField.Confidence
KfxImageFile	e.ImageFile
KfxValue	e.Value
KfxConfidence	e.Confidence
KfxSearchText	e.SearchText
KfxRegistrationHorizBMU	e.RegistrationHorizBMU
KfxRegistrationVertBMU	e.RegistrationVertBMU

Procedures and Functions

SBL	VB.NET
Function KfxLoadValidation (VerifyBatch As Integer, _ NumberOfDocsInBatch As Integer) As Integer	Private Sub Developer_Batch_Loading(sender As Object, e As BatchEventArgs) Handles Me.BatchLoading
Function KfxLoadValidation(VerifyBatch As Integer) As Integer	
Function KfxUnloadValidation () As Integer	Private Sub Developer_Batch_Unloading(sender As Object, e As BatchEventArgs) Handles Me.BatchUnloading
Function KfxDocPreProcess (Id As Long, NumberOfPages As Integer, AlreadyProcessed As Integer) As Integer	Private Sub Developer_Doc_Pre_Processing(sender As Object, e As PreDocumentEventArgs) Handles Me.DocumentPreProcessing
Function KfxDocPostProcess (Id As Long, DataAccepted As Integer) As Integer	Private Sub Developer_Doc_Post_Processing (sender As Object, e As PostDocumentEventArgs) Handles Me.DocumentPostProcessing
Function KfxFolderPreProcess(AlreadyProcessed As Integer) As Integer	Private Sub Developer_Folder_Pre_Processing(sender As Object, e As PreFolderEventArgs) Handles Me.FolderPreProcessing

SBL	VB.NET
Function KfxFolderPostProcess(DataAccepted As Integer) As Integer	Private Sub Developer_Folder_Post_Processing(sender As Object, e As PostFolderEventArgs) Handles Me.FolderPostProcessing
Function PreField_Name(Precision as Integer, Scale As Integer) As Integer	Private Sub Developer_Field_Name_Pre_Processing(sender As Object, e As PreFieldEventArgs) Handles Developer_Field_Name.FieldPreProcessing
Function PreField_Name() As Integer	
Function PostField_Name(EnteredValue As String, MaxLength As Integer) As Integer	Private Sub Developer_Field_Name_Post_Processing(sender As Object, e As PostFieldEventArgs) Handles Developer_Field_Name.FieldPostProcessing
Function PostField_Name(EnteredValue As String) As Integer	
Function PostField_Name(EnteredValue As String, Precision as Integer, Scale As Integer) As Integer	
Function FmtField_Name(Precision as Integer, Scale As Integer) As String	Private Sub Developer_Fld_Name_Formatting(sender As Object, e As FormatFieldEventArgs) Handles Developer_Field_Name.FieldFormatting
Function FmtField_Name() As String	
Function Pre_KFX_Field_Type_Name(Precision as Integer, Scale As Integer) As Integer	Private Sub Developer_Field_Type_Pre_Processing(s As Object, e As PreFieldEventArgs) Handles Me.FieldPreProcessing
Function Post_KFX_Field_Type_Name(EnteredValue As String, Precision as Integer, Scale As Integer) As Integer	Private Sub Developer_Field_Type_Post_Processing(s As Object, e As PostFieldEventArgs) Handles Me.FieldPostProcessing
Function Fmt_KFX_Field_Type_Name(Precision as Integer, Scale As Integer) As String	Private Sub Developer_Field_Type_Formatting(s As Object, e As FormatFieldEventArgs) Handles Me.FieldFormatting
Function KfxLoad() As Integer	Private Sub Developer_Recog_Profile_Loading(s As Object, ByRef ImageFileRequired As Boolean) Handles Me.BatchLoading
Function KfxPreRecognition As Integer	Private Sub Developer_Recog_Profile_Pre_Recognizing(s As Object, e As PreRecognitionEventArgs) Handles Me.RecognitionPreProcessing
Function KfxPostRecognition As Integer	Private Sub Developer_Recog_Profile_Post_Recognizing(s As Object, e As PostRecognitionEventArgs) Handles Me.RecognitionPostProcessing

With SBL validation script, you CANNOT declare a Sticky-and-Required index field variable without the field's post-processing function definition.

With VB.NET validation script, for an index field, you need to add a **handler** for each of the field's events to the Document/Folder Validation Script class if you want the field's processing/formatting functions will be called.

In SBL, all data types of index field variables are of the String data type.

In VB.NET, we can use arithmetic operators with a String as operand (see example below). So index field variables of field types *which are created from INTEGER, SMALLINT, DOUBLE, DECIMAL, NUMERIC, FLOAT and REAL base field types* are used very flexibly.

Note information: There is no equivalent for a SBL non-Tungsten-binding Global variable in Tungsten's support for VB.NET. For example, if we have two SBL document validation scripts loaded in one batch, with the script content as below, the variable *strAppModuleNameInWhichIam* will retain its value when validating documents belonging to these two document classes in a batch:

Document validation script 1:

```
Global strAppModuleNameInWhichIam As String
Function KfxLoadValidation ( VerifyBatch As Integer, _
    NumberOfDocsInBatch As Integer ) As Integer
    If strAppModuleNameInWhichIam = "" Then
        MsgBox("DocClass 1: The strAppModuleNameInWhichIam is """". It's gonna be set to
a new value.")
        If VerifyBatch = 0 Then
            strAppModuleNameInWhichIam = "Validation"
        Else
            strAppModuleNameInWhichIam = "Verify"
        End If
    Else
        MsgBox("DocClass 1: The strAppModuleNameInWhichIam has been set. It's value is
"" & strAppModuleNameInWhichIam & """.")
    End If
    KfxLoadValidation = NoError
    Exit Function
End Function
```

Document validation script 2:

```
Global strAppModuleNameInWhichIam As String
Function KfxLoadValidation ( VerifyBatch As Integer, _
    NumberOfDocsInBatch As Integer ) As Integer
    If strAppModuleNameInWhichIam = "" Then
        MsgBox("DocClass 2: The strAppModuleNameInWhichIam is """". It's gonna be set to
a new value.")
        If VerifyBatch = 0 Then
            strAppModuleNameInWhichIam = "Validation"
        Else
            strAppModuleNameInWhichIam = "Verify"
        End If
    Else
        MsgBox("DocClass 2: The strAppModuleNameInWhichIam has been set. It's value is
"" & strAppModuleNameInWhichIam & """.")
    End If
    KfxLoadValidation = NoError
    Exit Function
End Function
```

For more detail on what to do when converting script code from SBL to VB.NET, see the example below:

SBL script code:

```
Option Explicit On

Declare Function KfxValidateDecimal Lib "KfxValid.dll" _
    (DecimalString As String, ByVal Precision As Integer, _
    ByVal Scale As Integer) As Integer
```

```

Declare Function KfxFormatDecimal Lib "KfxValid.dll"
    (DecimalString As String, ByVal Precision As Integer, _
    ByVal Scale As Integer) As Integer
Declare Function KfxRoundInteger Lib "KfxValid.dll" (InValue As String, ByRef OutValue
    As Long) As Integer

Const RejectAndSkipDocument = -4
Const ValidationError = -3
Const ValidationErrorMessage = -2
Const FatalError = -1
Const NoError = 0
Const SaveAndSkipDocument = 1
Const SaveAndSkipField = 2
Const NoOperation = 3

Dim KfxBatchName As String
Dim KfxBatchId As String
Dim KfxBatchClassId As String
Dim KfxClassName As String
Dim KfxDocClassId As String
Dim KfxPageFile As String
Dim KfxErrorMessage As String
Dim KfxAcmDocument As Object
Global KfxOffers_Validator As String 'a Tungsten-binding variable for a Sticky-and-
Required index field named "Offers Validator"

Global strAppModuleNameInWhichIam As String

Function KfxLoadValidation(VerifyBatch As Integer, _
    NumberOfDocsInBatch As Integer) As Integer
    If strAppModuleNameInWhichIam = "" Then
        MsgBox("DocClass 1: The strAppModuleNameInWhichIam is "". It's gonna be set
to a new value.")
        If VerifyBatch = 0 Then
            strAppModuleNameInWhichIam = "Validation"
        Else
            strAppModuleNameInWhichIam = "Verify"
        End If
    Else
        MsgBox("DocClass 1: The strAppModuleNameInWhichIam has been set. It's value is
"" & strAppModuleNameInWhichIam & "".")
    End If
    KfxLoadValidation = NoError
    Exit Function
End Function

'---- When closing the batch
Function KfxUnloadValidation() As Integer
    MsgBox("DocClass 1: The " & strAppModuleNameInWhichIam & " script of the document
class "" & KfxClassName & "" (DocClassId = " & KfxDocClassId & ") has been unloaded
from the batch "" & KfxBatchName & "" (BatchId = " & KfxBatchId & ") of BatchClassId
" & KfxBatchClassId)
    KfxUnloadValidation = NoError
    Exit Function
End Function

Function KfxDocPreProcess(Id As Long, NumberOfPages As Integer, _
    AlreadyProcessed As Integer) As Integer
    If KfxOffers_Validator = "" Then
        'The document validation script has JUST been loaded when LOADING the batch OR
OPENING a document of ANOTHER document class
        MsgBox("DocClass 1: The Sticky KfxOffers_Validator variable has been reset")
    Else
        MsgBox("DocClass 1: The Sticky KfxOffers_Validator variable's value is "" &
KfxOffers_Validator & "")
    End If
End Function

```

```

End If
If AlreadyProcessed Then
    MsgBox("Already processed")
End If
KfxDocPreProcess = NoError
Exit Function
End Function

Function KfxDocPostProcess(Id As Long, DataAccepted As Integer) As Integer
    If DataAccepted = 1 Then
        MsgBox("Requested Save Index Data")
    Else
        MsgBox("Cancelled Save Index Data")
    End If
    KfxDocPostProcess = NoError
    Exit Function
End Function

Global KfxCustomer_Name As String

'---- We really DON'T NEED this function,
'---- because it does the same thing as the default behavior which occurs when NO post-
processing function for an index field of a String field type EXISTS
'---- The default behavior is to not accept overlength data.
Function PostCustomer_Name(EnteredValue As String, MaxLength As Integer) As Integer
    On Error GoTo Failure

    EnteredValue = Trim(EnteredValue)
    If (Len(EnteredValue) > MaxLength) Then GoTo Failure

    KfxCustomer_Name = EnteredValue
    PostCustomer_Name = NoError
    Exit Function
Failure:
    PostCustomer_Name = ValidationError
    Exit Function
End Function

Global KfxAssigned_Date As String
'---- Add 2 days to the Assigned Date index field value
Function PostAssigned_Date(EnteredValue As String) As Integer
    On Error GoTo Failure

    EnteredValue = Trim(EnteredValue)
    If Not IsDate(EnteredValue) Then GoTo Failure

    KfxAssigned_Date = CStr(DateValue(EnteredValue) + 2)
    PostAssigned_Date = NoError
    Exit Function
Failure:
    PostAssigned_Date = ValidationError
    Exit Function
End Function

Global KfxCLZip As String
Global KfxZip As String

Function PreZip() As Integer
    On Error GoTo Failure
    Dim SQL_INTEGER As Long
    If (KfxRoundInteger(KfxCLZip, SQL_INTEGER) <> NoError) Then
        GoTo Failure
    End If
    If SQL_INTEGER < 95 Then

```

```

        If KfxAcmDocument.Note = "" Then
            MsgBox("There's an error! We will see the page " & KfxPageFile & " later in
the Quality Control module.")
        Else
            MsgBox("We have processed this document in the Quality Control module with
a note "" & KfxAcmDocument.Note & """)
        End If
        'Do not assign a value to the KfxAcmDocument.Note attribute
        KfxErrorMessage = "The confidence level of the index field ""Zip"" is too low
(" & KfxCLZip & ")."
        GoTo Failure
    End If

    PreZip = NoError
    Exit Function
Failure:
    PreZip = FatalError
    Exit Function
End Function

'---- A post-processing validation behavior which is smarter
'---- than the default behavior which occurs when NO post-processing function for an
index field of an Integer field type EXISTS
'---- The default behavior is to not accept non-Integer data.
Function PostZip(EnteredValue As String) As Integer
    On Error GoTo Failure

    Dim SQL_INTEGER As Long
    EnteredValue = Trim(EnteredValue)
    If Not IsNumeric(EnteredValue) Then GoTo Failure

    If (KfxRoundInteger(EnteredValue, SQL_INTEGER) <> NoError) Then
        GoTo Failure
    End If

    KfxZip = Format(SQL_INTEGER, "0")
    PostZip = NoError
    Exit Function
Failure:
    PostZip = ValidationError
    Exit Function
End Function

'---- Zip format: add thousands symbol
'---- We really DON'T NEED this function,
'---- because it does the same thing as the default behavior which occurs when NO
formatting function for an index field of an Integer field type EXISTS
Function FmtZip() As String
    'The On Error is here to trap unexpected exceptions, however this function
    'does not return a status.
    On Error GoTo Failure
    Dim SQL_INTEGER As Long
    If (KfxZip = "") Then GoTo Failure

    If (KfxRoundInteger(KfxZip, SQL_INTEGER) <> NoError) Then
        GoTo Failure
    End If

    FmtZip = Format(SQL_INTEGER, "#,0")
    Exit Function
Failure:
    FmtZip = KfxZip
End Function

```

```

Global KfxOrder_Total As String

'---- In SBL, we CAN'T use arithmetic operators with a String as operand.
'---- We must convert the String to a number.
Function PreOrder_Total(Precision As Integer, Scale As Integer) As Integer
    On Error GoTo Failure
    KfxOrder_Total = CStr(CDbl(KfxOrder_Total) * 2)
    PreOrder_Total = NoError
    Exit Function
Failure:
    '---- probably couldn't find the DLL
    PreOrder_Total = FatalError
    Exit Function
End Function

'---- We really DON'T NEED this function,
'---- because it does the same thing as the default behavior which occurs when NO post-
processing function for an index field of a Decimal(10, 2) field type EXISTS
'---- The default behavior is to not accept non-Decimal(10, 2) data.
Function PostOrder_Total(EnteredValue As String, Precision As Integer, Scale As
Integer) As Integer
    On Error GoTo FatalFailure

    EnteredValue = Trim(EnteredValue)
    If (KfxValidateDecimal(EnteredValue, Precision, Scale) <> 0) Then GoTo Failure

    KfxOrder_Total = CStr(CDbl(EnteredValue) / 2)
    PostOrder_Total = NoError
    Exit Function
Failure:
    PostOrder_Total = ValidationError
    Exit Function
FatalFailure:
    '---- probably couldn't find the validation DLL
    PostOrder_Total = FatalError
    Exit Function
End Function

'---- Order_Total format: add thousands symbol
'---- We really DON'T NEED this function,
'---- because it does the same thing as the default behavior which occurs when NO
formatting function for an index field of a Decimal(10, 2) field type EXISTS
Function FmtOrder_Total(Precision As Integer, Scale As Integer) As String
    'The On Error is here to trap unexpected exceptions, however this function
    'does not return a status.
    On Error GoTo Failure
    Dim FormattedValue As String
    If (KfxOrder_Total = "") Then GoTo Failure
    FormattedValue = KfxOrder_Total
    If (KfxFormatDecimal(FormattedValue, Precision, Scale) <> 0) Then GoTo Failure
    FmtOrder_Total = FormattedValue
    Exit Function
Failure:
    FmtOrder_Total = KfxOrder_Total
End Function

'---- In SBL, we need this function because the "Offers Validator" index field is
Sticky and Required
Function PostOffers_Validator(EnteredValue As String, MaxLength As Integer) As Integer
    On Error GoTo Failure

    EnteredValue = Trim(EnteredValue)
    If (Len(EnteredValue) > MaxLength) Then GoTo Failure

```

```

    KfxOffers_Validator = EnteredValue
    PostOffers_Validator = NoError
    Exit Function
Failure:
    PostOffers_Validator = ValidationError
    Exit Function
End Function

```

VB .NET script code:

```

Imports Kofax.AscentCapture.NetScripting
Imports Kofax.Capture.CaptureModule.InteropServices
Imports System
Imports System.Collections.Generic
Imports System.Text
Imports System.Globalization 'for using CultureInfo

Namespace VBDotNetSharedNamespace

    <SuppressFieldEventsOnDocClose(False)> _
    Public Class VBDotNetForValidation
        Inherits DocumentValidationScript
        <IndexFieldVariableAttribute("Zip")> _
        Dim WithEvents Zip As FieldScript

        <IndexFieldVariableAttribute("Assigned Date")> _
        Dim WithEvents Assigned_Date As FieldScript

        '*** For Decimal data type (Precision and Scale) validation, we need just the
index-field-for-validation declaration only.
        '*** We don't need the index field's Post-processing procedure.
        '*** But we need the OrderTotal_Formating procedure.
        <IndexFieldVariableAttribute("Order Total")> _
        Dim WithEvents Order_Total As FieldScript

        '*** A Tungsten-binding variable for a Sticky-and-Required index field named
"Offers Validator"
        <IndexFieldVariableAttribute("Offers Validator")> _
        Dim WithEvents Offers_Validator As FieldScript

        '*** For String data type (length) validation, we need just the index-field-
for-validation declaration only.
        '*** We don't need the index field's Post-processing procedure.
        <IndexFieldVariableAttribute("Customer Name")> _
        Dim WithEvents Customer_Name As FieldScript

        Sub New(ByVal bIsValidatation As Boolean, ByVal strUserID As String, ByVal
strLocaleName As String)
            MyBase.New(bIsValidatation, strUserID, strLocaleName)
        End Sub

        Private Sub VBDotNetForValidation_BatchUnloading(sender As Object, e As
BatchEventArgs) Handles Me.BatchUnloading
            MsgBox("DocClass 1: The " & Me.Application.ToString & " script of
the document class "" & Me.Offers_Validator.IndexField.Document.ClassName &
"" (DocClassId = " & Me.Offers_Validator.IndexField.Document.ClassID & ") has been
unloaded from the batch "" & e.Batch.Name & "" (BatchId = " & e.Batch.ExternalID &
") of BatchClassId " & e.Batch.ClassID)
        End Sub

        Private Sub VBDotNetForValidation_DocPreProcessing(sender As Object, e As
PreDocumentEventArgs) Handles Me.DocumentPreProcessing

```

```

        AddHandler Assigned_Date.FieldPostProcessing, AddressOf
AssignedDate_PostProcessing
        AddHandler Zip.FieldPreProcessing, AddressOf Zip_PreProcessing
        AddHandler Zip.FieldPostProcessing, AddressOf Zip_PostProcessing
        AddHandler Zip.FieldFormatting, AddressOf Zip_Formatting
        AddHandler Order_Total.FieldPreProcessing, AddressOf
OrderTotal_PreProcessing
        AddHandler Order_Total.FieldFormatting, AddressOf OrderTotal_Formating

        If Offers_Validator.IndexField.Value = "" Then
            'The document validation script has JUST been loaded when LOADING the
batch OR OPENING a document of ANOTHER document class
            MsgBox("DocClass 1: The Sticky Offers_Validator variable has been
reset")
        Else
            MsgBox("DocClass 1: The Sticky Offers_Validator variable's value is ""
& Offers_Validator.IndexField.Value & """)
        End If
        If e.AlreadyProcessed Then
            MsgBox("Already processed")
        End If
    End Sub
    Private Sub VBDotNetForValidation_DocPostProcessing(sender As Object, e As
PostDocumentEventArgs) Handles Me.DocumentPostProcessing
        If e.DataAccepted Then
            MsgBox("Requested Save Index Data")
        Else
            MsgBox("Cancelled Save Index Data")
        End If
    End Sub
    '*** Add 2 days to the Assigned Date index field value
    Private Sub AssignedDate_PostProcessing(sender As Object, e As
PostFieldEventArgs) Handles Assigned_Date.FieldPostProcessing
        Dim dateTemp As Date
        dateTemp = Date.Parse(Assigned_Date.IndexField.Value).AddDays(2)
        Assigned_Date.IndexField.Value = dateTemp.ToShortDateString
    End Sub

    '*** We can use arithmetic operators with a String as operand
    Private Sub OrderTotal_PreProcessing(sender As Object, e As PreFieldEventArgs)
Handles Order_Total.FieldPreProcessing
        Order_Total.IndexField.Value = Order_Total.IndexField.Value * 2
    End Sub

    Private Sub OrderTotal_Formating(sender As Object, e As FormatFieldEventArgs)
Handles Order_Total.FieldFormatting
        ScriptUtils.KfxFormatDecimal(e.DisplayValue,
Order_Total.IndexField.NumericPrecision, Order_Total.IndexField.NumericScale)
    End Sub

    Private Sub Zip_PreProcessing(sender As Object, e As PreFieldEventArgs) Handles
Zip.FieldPreProcessing
        If Zip.IndexField.Confidence < 95 Then
            If Zip.IndexField.Document.Note = "" Then
                MsgBox("There's an error! We are seeing the page " &
Zip.IndexField.Page.FileName & " later in the Quality Control module.")
            Else
                MsgBox("We have processed this document in the Quality Control
module with a note "" & Zip.IndexField.Document.Note & """)
            End If
            'Do not assign a value to the Zip.IndexField.Document.Note attribute
            'This is just a validation error message!
            Throw New RejectAndSkipDocumentException("The confidence level of the
index field ""Zip"" is too low (" & Zip.IndexField.Confidence & ").")
        End If
    End Sub

```

```
        End If
    End Sub

    '*** We need this Post-processing procedure to check if the input data is
    numeric, e.g: 10761.94 => "$10,761.94" or "1.076E+004"
    Private Sub Zip_PostProcessing(sender As Object, e As PostFieldEventArgs)
    Handles Zip.FieldPostProcessing
        Dim nZip As Integer
        If Not IsNumeric(Zip.IndexField.Value) Then
            Throw New RejectAndSkipDocumentException("The input data "" &
Zip.IndexField.Value & "" is not numeric!")
        End If
        If ScriptUtils.KfxRoundInteger(Zip.IndexField.Value, nZip) = False Then
            Throw New RejectAndSkipDocumentException("The input data "" &
Zip.IndexField.Value & "" can not be rounded!")
        End If
        Zip.IndexField.Value = Format(nZip, "0")
    End Sub

    Private Sub Zip_Formatting(sender As Object, e As FormatFieldEventArgs) Handles
Zip.FieldFormatting
        Dim nZip As Integer
        nZip = Zip.IndexField.Value
        e.DisplayValue = nZip.ToString("N0", CultureInfo.InvariantCulture)
    End Sub
End Class
End Namespace
```