

# **Kofax Transformation Modules**

5.0

Configuration Guide



© 2006-2010 Kofax, Inc. Portions © 2002-2006 Kofax Development GmbH. Portions © 1997-2006 Kofax U.K., Ltd. All rights reserved. U.S. Patent No. 5,159,667. Use is subject to license terms. Kofax and the Kofax Logo are registered trademarks of Kofax, Inc. All product names are the trademarks or registered trademarks of Kofax, Inc. or its suppliers.

---

# Contents

<b>Overview .....</b>	<b>6</b>
<b>Script Editor's User Interface .....</b>	<b>7</b>
<b>Getting Started .....</b>	<b>9</b>
XDocument .....	9
Object Model of the XDocument Object .....	10
Object Model of the XDocField Object .....	11
XFolder and XDocInfo .....	11
Object Model of the XFolder and XDocInfo Objects .....	12
XValue .....	12
Application .....	13
Batch .....	15
Batch_Close Event, CloseMode Parameter .....	16
RootFolder .....	16
ValidationForm and ValidationPanel .....	17
Object Model of ValidationForm Object .....	18
Object Model of ValidationPanel Object .....	18
VerificationForm and VerificationPanel .....	19
Object Model of VerificationForm Object .....	20
Object Model of VerificationPanel Object .....	20
Project .....	20
Scripting Logic .....	20
Project Script .....	21
Class Script .....	22
RootFolder Script .....	23
Folder Script .....	24
Inserting Script Events .....	25
<b>Server Events Scripting .....</b>	<b>30</b>
Batch Processing .....	31
Parallelization .....	33
Document Separation .....	33
Standard Document Separation .....	33
Trainable Document Separation (TDS) .....	34
Document Processing .....	35
Processing Project Fields .....	36
Classification .....	37
Extraction .....	39
Validation .....	43
Folder Extraction and Validation .....	43

<b>Document Review Events Scripting .....</b>	<b>47</b>
Document Review Methods .....	47
Batch Editing Events .....	47
Events for Document Review's Customized Menu Commands .....	48
<b>Validation Module Events Scripting .....</b>	<b>50</b>
Validating Fields .....	50
Validation Layout Events .....	50
Events of the Field Control .....	51
Events of the Table Control .....	52
Events of the Tab Control .....	52
Events of the Button Control .....	52
Batch Editing Events .....	52
<b>Verification Module Events Scripting .....</b>	<b>53</b>
Verification Layout Events .....	53
<b>Applying Scripting .....</b>	<b>54</b>
Formatting .....	54
Validating .....	54
Server Module Validation Events .....	55
ValidationMethod Events .....	55
Access to Kofax Capture Data .....	56
Batch Data .....	57
Folder Data .....	58
Document Data .....	58
Foldering .....	60
Summarizing Field Data .....	62
Changing Folder and Document Field Data .....	64
Batch Restructuring .....	68
Page Operations .....	68
Document Operations .....	70
Folder Operations .....	72
Document Routing .....	74
Moving Documents to a child Batch .....	75
Assigning a Batch to Modules in the Queue .....	76
Batch Editing .....	77
Adding Page .....	78
Deleting Page .....	79
Moving Page .....	81
Rotating Page .....	82
Adding Document .....	83
Before Overriding Document Problem .....	85
Before Restoring Document Problem .....	86
Changing Class .....	87
Confirming Class .....	88
Copying Document .....	89
Creating Document .....	91
Deleting Document .....	92

Merging Document .....	94
Moving Document .....	95
Splitting Document .....	96
Before Overriding Folder Problem .....	97
Before Restoring Folder Problem .....	98
Creating Folder .....	98
Deleting Folder .....	100
Merging Folder .....	102
Moving Folder .....	103
Splitting Folder .....	104
Line Item Matching Locator .....	106
Using Match Function In Validation .....	107
Using Match Function During Server Processing .....	108
<b>Script Samples .....</b>	<b>109</b>
Set a Field's Confidence Threshold from a Script Variable .....	109
Skip Layout Classification After Second Page .....	109
Reclassify if the classification result has a specific value .....	110
Find most recent date .....	110
Dynamically Suppress Orientation Detection for Full Page OCR .....	111
Add an Alternative .....	111
Classification by Blackness for Single Classes .....	112
Classification by Graphical Lines .....	113
Accepting An Empty Field .....	114
Formatting Negative Amounts .....	115
Access DPI .....	115
Suppress OCR After Third Page .....	116
Use Project Script Execution Mode .....	116
Accessing Preprocessed Zones Images of Advanced Zone Locator .....	116
Turn off Statistics .....	117
Getting the Root XFolder Object by Given XDocument .....	117
<b>Using Help .....</b>	<b>118</b>
Navigating the Help .....	118
Contents .....	118
Glossary .....	118
Index .....	119
Search .....	119
Help Topics .....	119
Procedures .....	119
Examples and Other Collapsed Text .....	120
Notes, Tips, Important, and Caution .....	120
<b>Related Documentation .....</b>	<b>121</b>
<b>Technical Assistance for Your Kofax Product .....</b>	<b>122</b>
<b>Glossary .....</b>	<b>123</b>

---

# Overview

The WinWrap Basic Editor is a VBA-compatible script engine that can be used to enhance the built-in Kofax Transformation Modules' classification, extraction, and validation processes. It is an interactive design environment for developing, testing, and executing WinWrap Basic scripts. Events are fired that can be used to individually program new classification and/or extraction or validation procedures.

Besides a short introduction to the editor's user interface, this documentation provides an overview of the available events. Small code samples show their usage, and figures illustrate the event sequence during the document recognition process.

In the script environment, the document model of the CASCADE component suite objects is available for the experienced script programmer. To improve the usability of the COM components, an additional *Kofax Transformation Scripting Object References Help* is provided that contains the API documentation for the relevant object libraries.

The topics within this document are arranged thusly:

- [Getting Started](#) - an introduction to the and to the and scripting objects.
- [Server](#), [Document Review](#), [Validation](#), [Verification](#): Module specific scripting (by module that is supporting scripting) - these topics cover the scripts that have relation to a specific module.
- [Applying Scripting](#) - this topic is arranged by function and under certain conditions with information about module specific implementation.
- [Code Snippets](#) - this section contains code examples for the common scripting requirements.

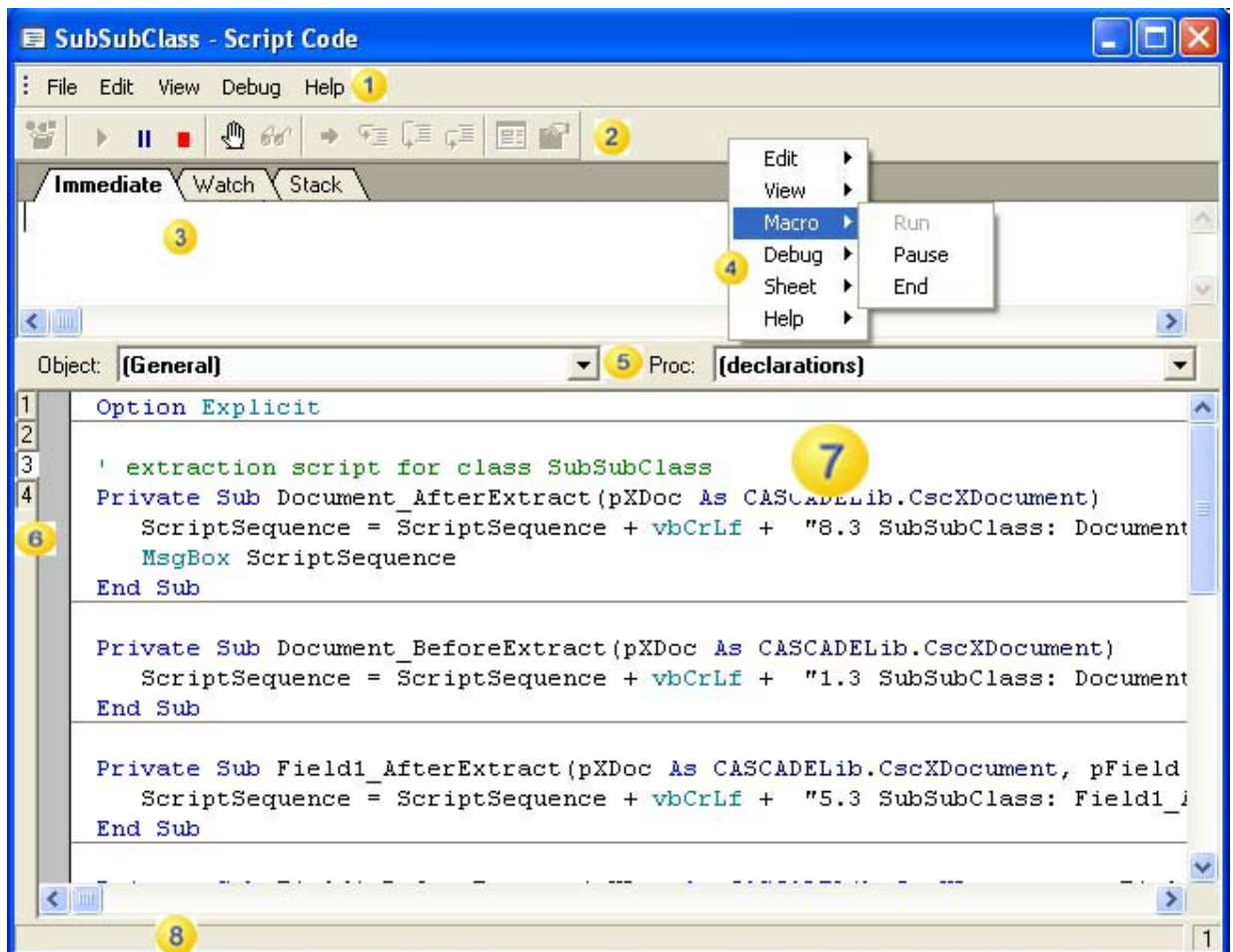
**Please note that only those objects, methods and properties that are documented in the API documentation can be used for script programming. The use of undocumented methods or properties in script is not recommended and not supported and can cause unpredictable behavior.**

# Script Editor's User Interface

This topic provides an introduction to the user interface for the script editor.

To open the script editor, do one of the following:

- From the View menu select Show Script.
- Choose Show Script from the projects or class context menu.
- Click the Show Script icon in the toolbar.



## WinWrap Basic Editor User Interface

1

### Menu Bar

The script editor supports a standard, Windows-style menu bar from which you can perform various operations.

2

**Toolbar**

The toolbar provides shortcuts to many menu items. The following table provides an overview of the available buttons.

3

**Immediate / Watch / Stack**

Additional windows are used during debugging to list variables, functions, or expressions that are calculated and displayed. For further information, see the script editor's Help.

4

**Context Menus**

Context menus are available. Right-click anywhere in the script editor to display a context menu. For further information, see the script editor's Help.

5

**Object / Proc drop-down list boxes**

The Object / Proc drop-down list boxes show the objects and procedures that are available for the selected class.

Select an item from the Object drop-down list box (for example, "Document") to implement a document extraction event. To implement a field extraction event, select <Field>, where <Field> is the name of the extraction field. To implement a locator method for a field extraction, select <Locator>, where <Locator> is the name of the locator field.

From the Proc drop-down list box, choose the event you want to implement. For example, select "Field1" from the Object drop-down list box and `After_Extract` from the Proc drop-down list box. This inserts the `Field1_AfterExtract` event into the selected sheet. If you want to insert a classification event, you must change to the project level. To do this, first switch to sheet 1 and select "Document" from the Object drop-down list box and, for example, `AfterClassifyImage` from the Proc drop-down list box to insert the `Document_AfterClassifyImage` event. For further information concerning the user interface, see the script editor's Help.

6

**Sheets Tab**

For each class and for each folder for which you show the script, a sheet is provided in the script editor that contains the script code for the class or the folder. The project script is always in the first script sheet. The editor can show up to ten different sheets. The first sheet always shows the script code on the project level, on which you implement the classification events. The other sheet tabs can be used to display the script code for all classes of the project, containing the events that occur during the extraction and for the script locator. Click on a sheet tab to show the script code area for a class. Double-click the tab to close the script code area for the class.

7

**Edit Area**

The script code for the selected class can be viewed or edited in this area.

8

**Status Bar**

The status bar provides status messages and further information.



---

# Getting Started

There are various instances within each document recognition process where the option to customize its behavior can be quite useful. During document separation, classification, extraction, validation, as well as foldering and verification, various events are fired that can be filled with customizing script code. The WinWrap Basic Script editor is used to develop and test WinWrap Basic scripts that are executed for the available events by the WinWrap script engine.

- [XDocument](#) - CscXDocument
- [XFolder and XDocInfo](#) - CscXFolder and CscXDocInfo
- [XValue](#) - CscXValue
- [Project](#) - CscProject
- [Application](#) - CscScriptApplication
- [Batch](#) - CscScriptBatch
- [ValidationForm and ValidationPanel](#) - CscScriptValidationForm and CscScriptValidationPanel
- [VerificationForm and VerificationPanel](#) - CscScriptVerificationForm and CscScriptVerificationPanel

**Please note that only those objects, methods and properties that are documented in the API reference can be used for script programming. The use of undocumented methods or properties in script is not recommended and not supported and can cause unpredictable behavior.**

## XDocument

Within Kofax Transformation Modules, the extraction of field data is based on an internal representation of the document called the XDocument, which contains all of the layout elements of the document. Layout elements are text elements (words and their geometries, fonts, and location) and graphical elements (lines, logos, and textures). This internal representation is used by the extraction algorithms throughout the complete process.

The XDocument (CscXDocument) object is part of the CscXDocument library and is the core object for representing document information. The XDocument is created during OCR for scanned documents or by a conversion filter for electronic documents (for example, PDF). This standardized representation ensures format independence for the entire process. The XDocument also functions as a container for extraction and analysis results to be used in further processing steps.

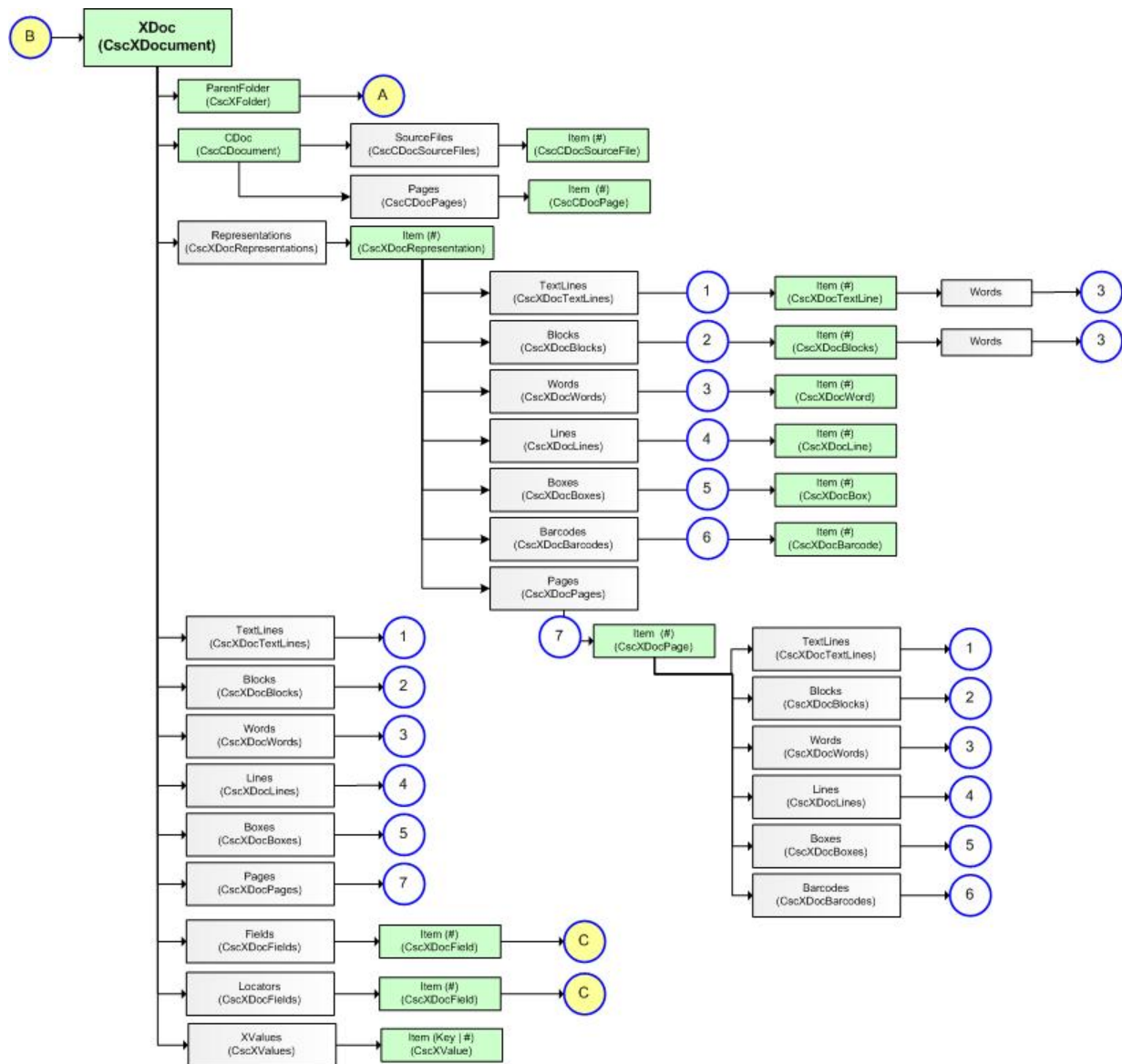
The XDocument represents an array of logical structures that make up the source file(s). This means that it can contain a representation of several TIFF images on which OCR can be performed. The XDocument thus consists of a representation that contains pages, text lines, and words. If another OCR with different settings is performed, the XDocument contains a second representation that can have a different structure of lines and words. Basically, an XDocument is used to store all of the derived information about a file (or several files) from its creation through all kinds of analysis (for example, OCR and line analysis), classification, and data extraction, up to export or final archiving.

The XDocument consists of a collection of representations and a CDocument that defines the page structure of all representations and provides information about the source files. The CDocument represents a compound document. You can define a number of source files or even plain text snippets that you want to use to construct an XDocument. The files can be of different types (for example, TEXT, PDF, TIFF, or Microsoft Word). Whenever a source file is added to the CDocument, its page structure is analyzed, and a Page is created. All of those pages together define the page structure for all representations in an XDocument.

[Library Model of the XDocument Object](#)

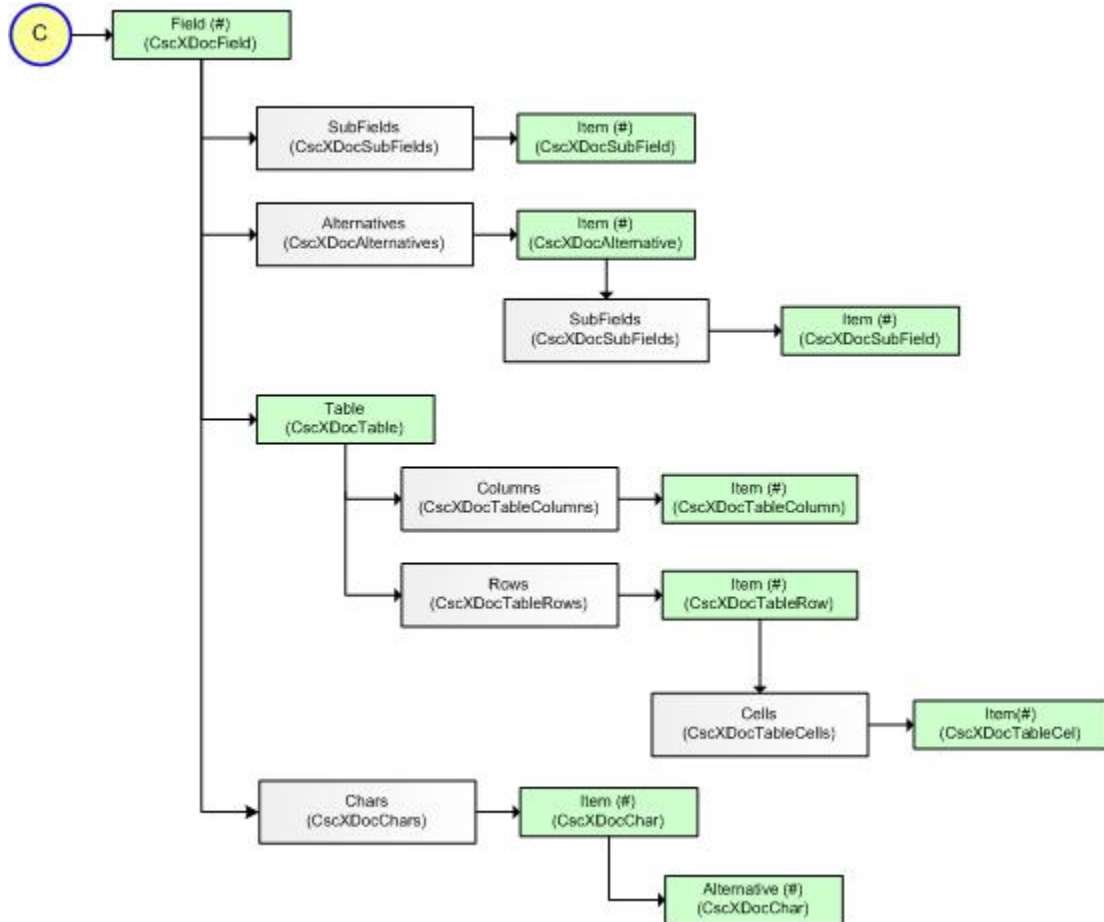
[Library Model of the XDocField Object](#)

## Object Model of the XDocument Object



**Library Model CscXDocument Object**

## Object Model of the XDocField Object



### Library Model CscXDocField Object

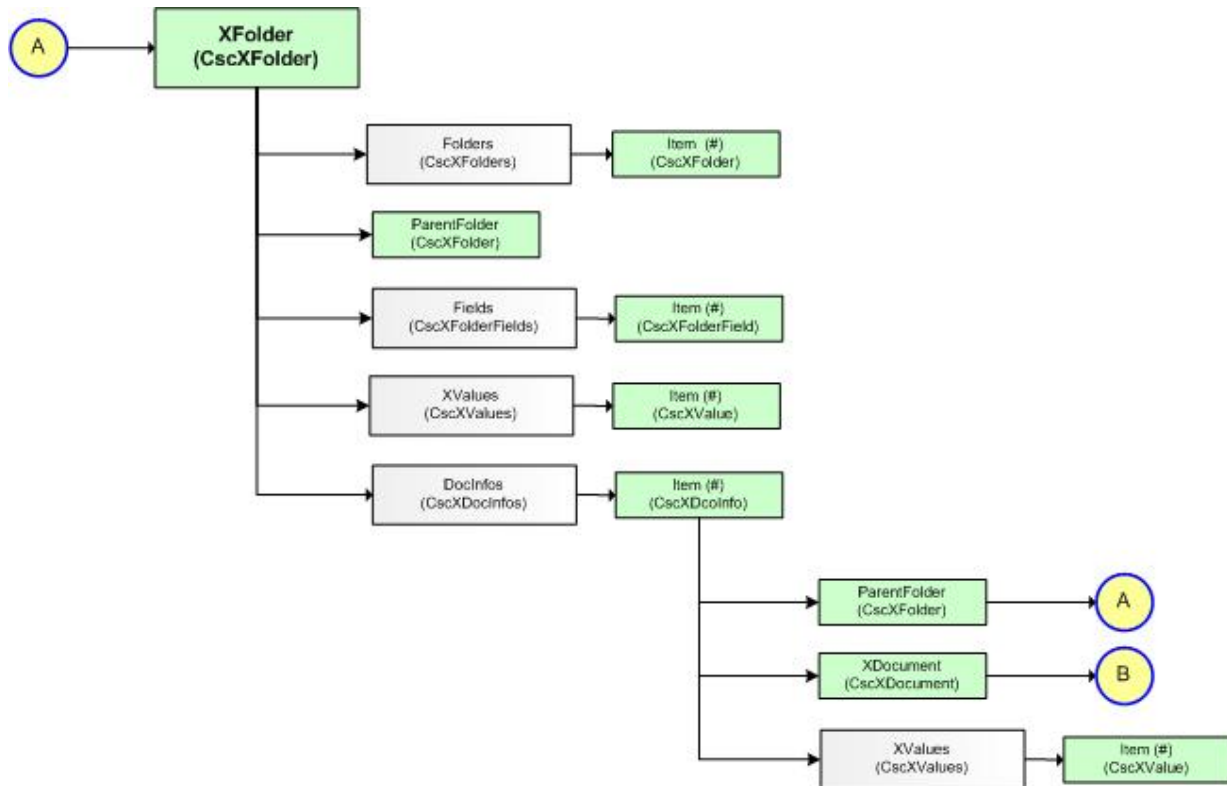
## XFolder and XDocInfo

With the introduction of foldering an object has been added to represent the batch structure, the XFolder (CscXFolder) object. The root object represents the batch and reflects its structure containing all dependent sub folders and documents in hierarchically organized folder and document collections. It is typically called root XFolder. The [XDocument](#) is represented by a new object called XDocInfo (CscXDocInfo) that allows basic access to core information of the document without loading the XDocument object itself what might be an expensive operation. An object corresponding to the XDocument object for an XFolder does not exist, the XFolder is fully represented by the XFolder object.

These objects are present independent of foldering. If foldering is not activated the root XFolder does not contain any sub folders and it is not possible to handle batch level fields, but the management of the documents like the definition of the sequence is done by its collection of XDocInfos.

[Library Model of the XFolder/XDocInfo Objects](#)

## Object Model of the XFolder and XDocInfo Objects



### Library Model CscXFolder Object

## XValue

The XValues (CscXValues) collection of an [XFolder](#) or an [XDocInfo](#) / [XDocument](#) object is a collection that supports key / value pairs. It can be accessed by index or by key.

XValues can be used to attach specific information to these objects. The XValues of the XDocInfo / XDocument objects are identical and synchronized; for the scripting user there is not difference from which object they are accessed.

The XValue (CscXValue) object is saved with the root XFolder object by setting the property 'MustSave' to true (default is true), so by default all created XValues are saved. If setting it to 'false' the XValue exists as long as the root XFolder remains loaded. Take into consideration for the accessibility of XValues during server processing that the root XFolder gets loaded/unloaded several times by different processes.

For accessing the XValues the following properties and methods are available:

- Checking if an XValue exists:

```
If MyObject.XValues.ItemExists("XValue_Key") Then
...
End If
```

- Accessing the value of an existing XValue uses the following syntax (before doing so you can check for its existence):

```
Dim sValue As String
```

```
sValue = MyObject.XValues("XValue_Key")
```

- Iterating over all XValue objects in the XValues collection:

```
Dim oXValue As CascadeLib.CscXValue

For i As Integer = 0 To MyObject.XValues.Count - 1
    oXValue = MyObject.XValues.ItemByIndex(i)
    ...
Next
```

- Setting the value of an XValue object. The 'Set' method implicitly creates an XValue with the given key if it does not exist and sets the given value. By default the 'MustSave' property is set to 'true':

```
MyObject.XValues.Set("XValue_Key", "XValue_Value")

' setting the new created or updated XValue
' to .MustSave = False
Dim oXValue As CascadeLib.CscXValue
Set oXValue = MyObject.XValues.ItemByName("XValue_Key")

oXValue.MustSave = False
```

- Getting the XValue object by name to update its value and setting the MustSave property to False so the XValue gets destroying when the root XFolder gets unloaded.

- Dim oXValue As CascadeLib.CscXValue
 

```
Set oXValue = MyObject.XValues.ItemByName("XValue_Key")

oXValue.Value = "New content for the value."

oXValue.MustSave = False
```

- Adding a new XValue object that is not saved with the root XFolder:

```
MyObject.XValues.Add("NewXValue_Key", "NewXValue_Value", false)
```

- Deleting an XValue object:

```
MyObject.XValues.Delete("XValue_Key")
```

- Clearing all XValue objects from the collection:

```
MyObject.XValues.Clear()
```

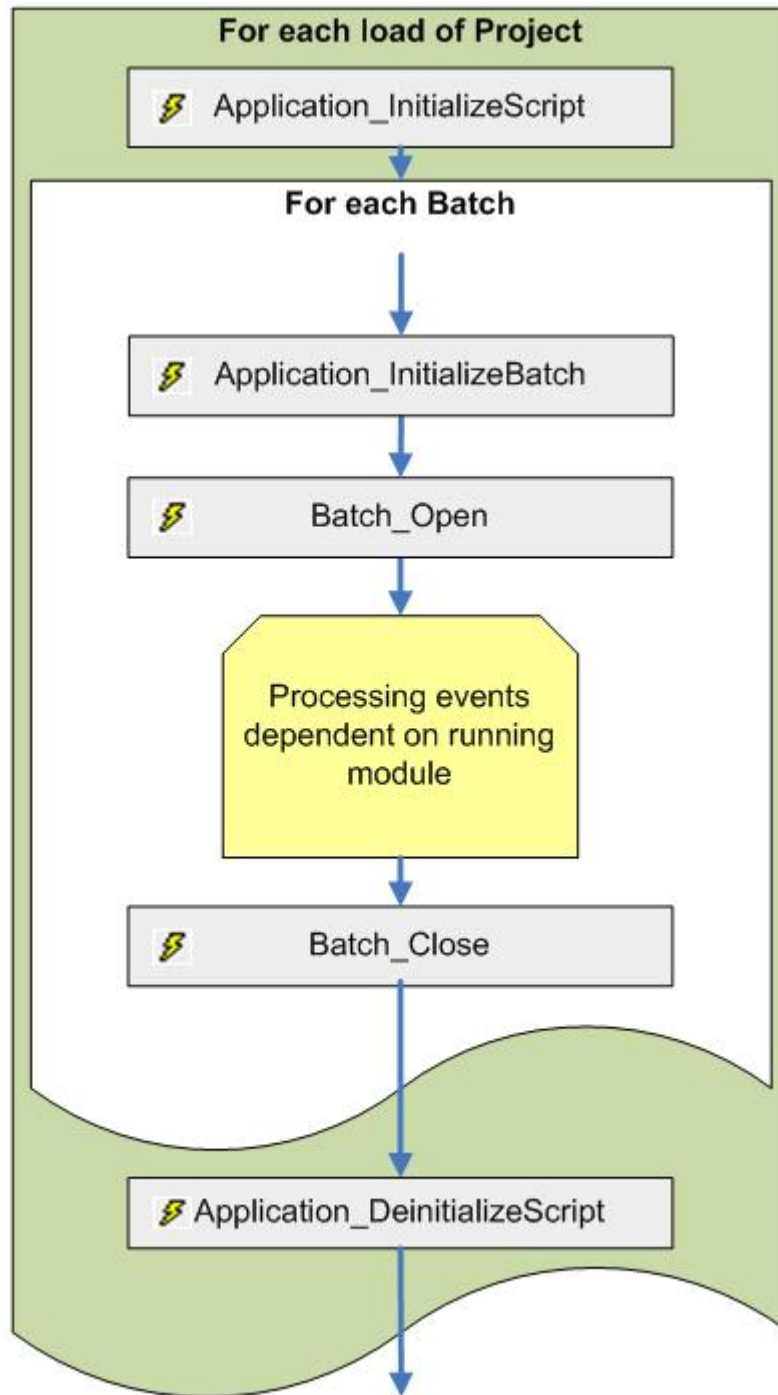
TODO add here a link to functional scripting using XValues or so?

## Application

The Application (CscScriptApplication) object is the object in script to that all application related events and properties are attached. It is placed in the project script sheet and can be addressed by Application.

The events can be divided into events that are available for every module that supports scripting and those that are only related to user interaction in Document Review.

The events that are available for every module are shown the following picture.



### Application Event Sequence

The `Application_InitializeScript` and the `Application_DeinitializeScript` event are bracketing the events that are controlling the batch processing. These events are related to the load of a new [Project](#) object and the initialization of the script engine. Typically this event is designed for the initialization of variables with a global scope. The deinitialization of the script engine gets announced by the `Application_DeinitializeScript` event so that it can be used for freeing any global variables.

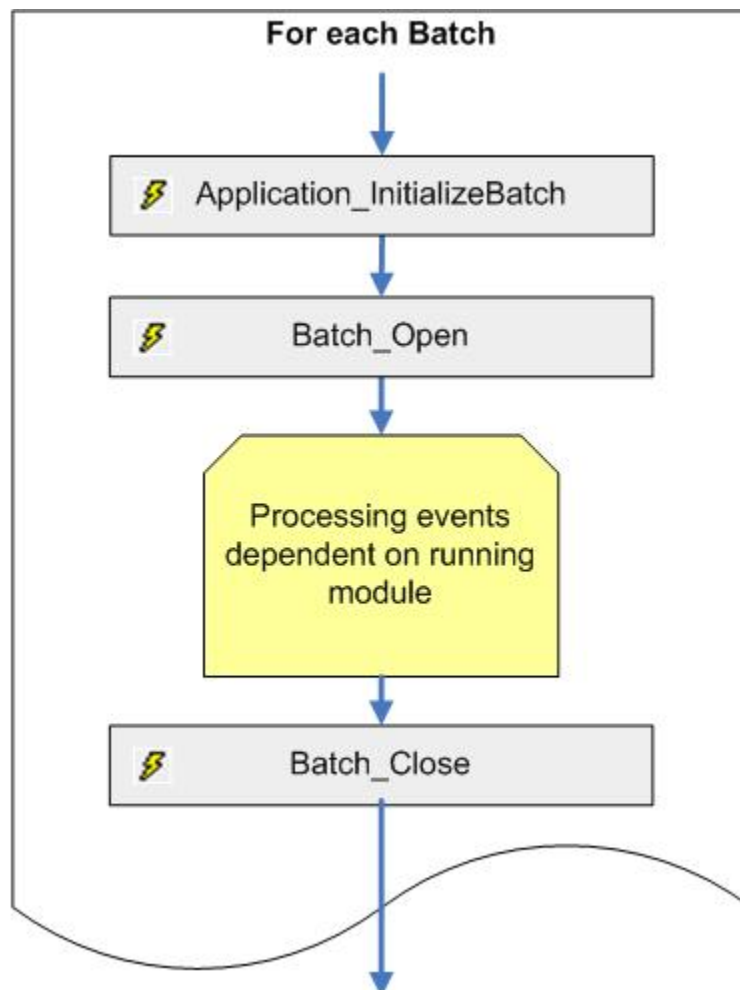
The event `Application_InitializeBatch` is announcing the processing of a new batch. In case of the user modules this is a deterministic sequence but for server processing this event always introduces the processing of a new or a different batch. For further details please refer to [Batch Processing](#) in server.

## Batch

The Batch (`CscScriptBatch`) object is the core object in script to that all batch related events and batch related operations like [Batch Restructuring](#) are attached. It is placed in the project and the root folder script sheet and can be addressed by `Batch`.

The events of the Batch object are available in the project script sheet and can be divided into events that are available for every module that supports scripting and those that are only related to user interaction during batch editing.

In the following the events are described that are available for every module and that allow a customization of the batch handling.



### Batch Event Sequence

This sequence can be expected for every user interactive module. The first event is not an event that is attached to the Batch object but is initializing its processing, it is attached to the `Application` object that is also defined on the project script sheet. In the `Application_InitializeBatch` event the script has read-only access to the given root `XFolder` object. In those events no changes to the batch



structure or to the root XFolder are allowed, but initializations that are batch related can be applied and batch properties can be read.

The `Batch_Open` and the `Batch_Close` events are thrown only once per modul and per batch and provide an extended access to the batch object that allows [Batch Restructuring](#) or [Document Routing](#) (only in `Batch_Close`), this is also true for the parallelized server processing. Whereas the `Application_InitializeBatch` event is thrown multiple times dependent of the number of server threads that are running independently on that batch.

The `Batch_Close` event provides a new parameter that informs about the close mode of the batch that is possible to differ between suspending, finally closing or closing in error.

To distinguish the executing module in these shared events please use the property `Project.ScriptExecutionMode` and if applicable the property `Project.ScriptExecutionInstance` for getting the information about the running instance of the module.

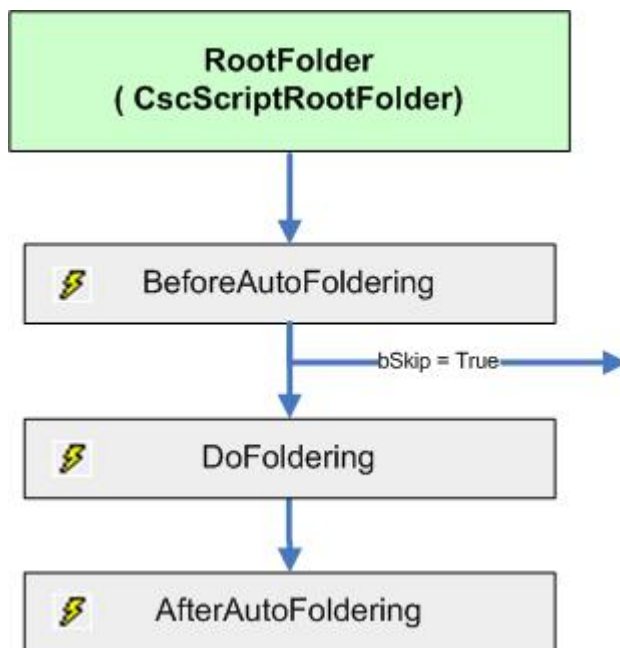
### Batch\_Close Event, CloseMode Parameter

When a batch gets closed in any module the `Batch_Close` event is fired once per batch. A batch is closed by finally closing it suspending, in error mode or finally closing it. This closing status is reflected in the `CloseMode` parameter that is passed to the `Batch_Close` event.

## RootFolder

The `RootFolder` (`CscScriptRootFolder`) object is only available if folders are enabled. It provides events for foldering.

Foldering events are only available if folders are enabled for the project and occur only during server processing or if allowed during validation manually initiated. You can distinguish between server processing and manual initiation by checking the property `Project.ScriptExecutionMode`. The following figure illustrates the script event sequence during foldering:



### Foldering Event Chronology

The `BeforeAutoFoldering` and `AfterAutoFoldering` events are already fired but not yet supported by a configuration interface. In the `DoFoldering` event, the batch can be restructured as



shown in [Folder Extraction and Validation Events](#). The complete foldering process can be skipped by setting the `bSkip` parameter of the `BeforeAutoFoldering` event to `TRUE`.

## ValidationForm and ValidationPanel

The `ValidationForm` (`CscScriptValidationForm`) object contains the layout definition of a validation form. This object is placed on the scripting sheet of a class. It is available if a validation form has been defined for that class. It is representing the document content.

If `enable folders` is activated in the project settings the corresponding object that reflects the folder content is the `ValidationPanel` (`CscScriptValidationPanel`).

The following graphic shows the Validation runtime with the document and the folder field area:

The screenshot displays a validation interface with two main sections: **ValidationPanel (Folder Data)** and **ValidationForm (Document Data)**.

**ValidationPanel (Folder Data):** This section shows the classification result and validation status of fields. It includes a dropdown menu for the class (set to 'Base') and a summary of 4 valid and 2 invalid fields. The fields are:

Field Name	Status	Value
BatchTotal	Valid (Green Checkmark)	3709,50
SupplierTotal	Valid (Green Checkmark)	1595,60

An **Update** button is located next to the SupplierTotal field.

**ValidationForm (Document Data):** This section shows the document data fields. It includes:

Field Name	Value	Status
InvoiceNumber	1401	Invalid (Red 'I' icon)
SupplierID	56789	Valid (Green Checkmark)
Total	1.445,60	Invalid (Red 'I' icon)

The `ValidationForm` object provides access to the defined controls. The layout of an `ValidationForm` can consist of one or more tabs. If only one tab is defined it is not displayed in validation runtime; thus the access to the first element of the tabs collection is then ignored.

On a `ValidationForm` you can define buttons, labels, fields, mini viewers, tables and groups on a tab. The collections of the `Tabs`, `MiniViewers`, `Labels`, `Fields`, `Buttons`, `Tables` and `Groups` of the `ValidationForm` object contain all controls, independent of the tabs or groups in which they are defined. The collections of a tab contain all controls that are placed directly on the tab (`MiniViewers`, `Labels`, `Fields`, `Buttons`, `Tables` and `Groups`). `MiniViewers`, `Labels`, `Fields` and `Buttons` can be grouped so that these controls become part of the collections of the group object.

The `ValidationPanel` object is part of the folder script and analogous to the `ValidationForm` object it is available when a form layout has been defined for that folder. The `ValidationPanel` object provides access to all defined folder field related controls on its layout. Buttons, labels, fields and group can be defined and accessed through the collections of the `Buttons`, `Labels`, `Fields` and `Groups`. As for the `ValidationForm` object all controls can be accessed by the collection objects of the `ValidationPanel` object independent of the group they are defined in. If `Fields`, `Labels` and `Buttons` are grouped the group object contains them additionally.

If a `ValidationPanel` is available in script it provides access by its property `ValidationForm` to the `ValidationForm` object that is displayed at the same time. The same is true for the `ValidationForm` object it provides access via property `ValidationPanel` to the folder field layout.

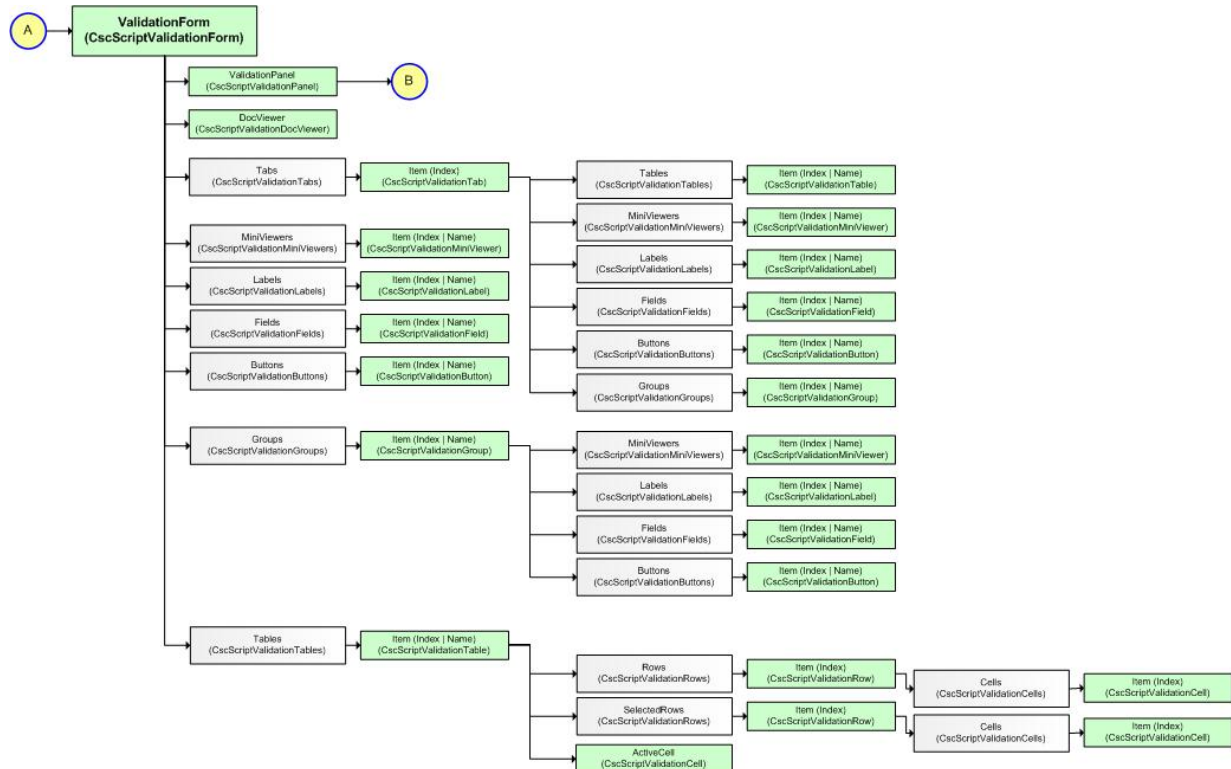
For a detailed reference of the available methods and properties of both objects please refer to the API reference and to the corresponding object model:

[ValidationForm and ValidationPanel Events](#)

[Object Model of ValidationForm \(Document Data\)](#)

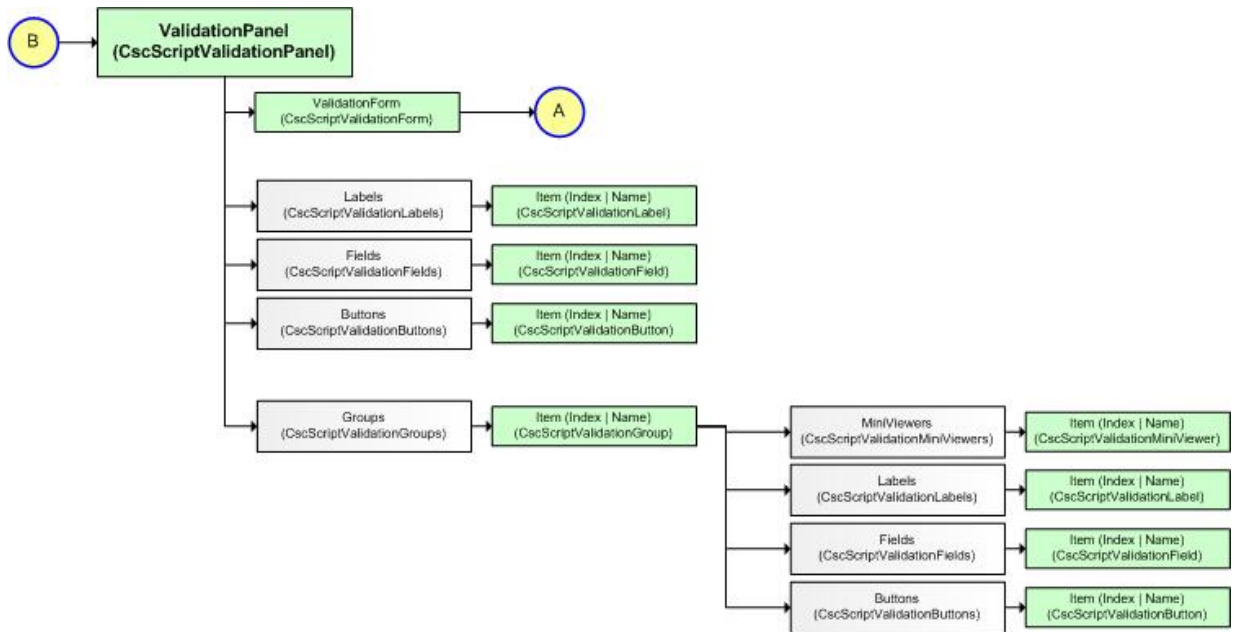
[Object Model of ValidationPanel \(Folder Data\)](#)

## Object Model of ValidationForm Object



## Object Model ValidationForm (Document Data)

## Object Model of ValidationPanel Object



### Object Model ValidationPanel (Folder Data)

## VerificationForm and VerificationPanel

The VerificationForm (CscScriptVerificationForm) contains the default layout definition of a verification form. This object is placed on the scripting sheet of a class. It is only available if at least one field of the class has been configured for verification.

The following graphic shows the Verification runtime with the document and the folder field area:



The VerificationForm object provides access to the by default generated labels for the fields that have been activated for verification and the document viewer.

The VerificationPanel object is part of the folder script and analogous to the VerificationForm object it is available when at least one folder field has been activated for verification. The VerificationPanel object provides access to the labels for all folder fields to be verified.

If a VerificationPanel is available in script it provides access by its property `VerificationForm` to the VerificationForm object that is displayed at the same time. The same is true for the VerificationForm object it provides access via property `VerificationPanel` to the folder field layout.

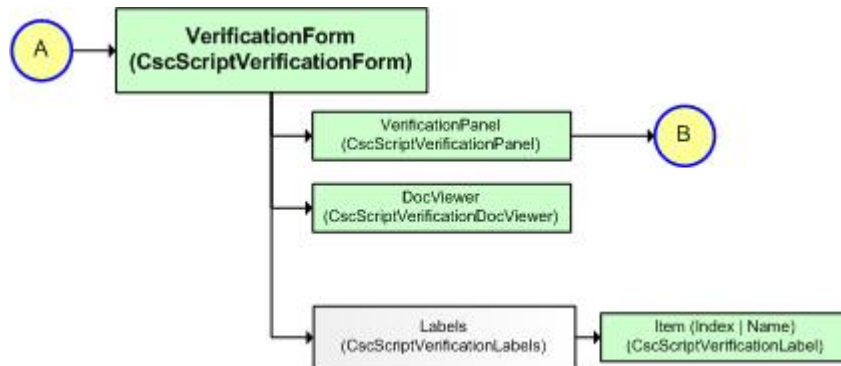
For a detailed reference of the available methods and properties of both objects please refer to the API reference and to the corresponding object model:

VerificationForm and VerificationPanel Events

[Object Model of VerificationForm \(Document Data\)](#)

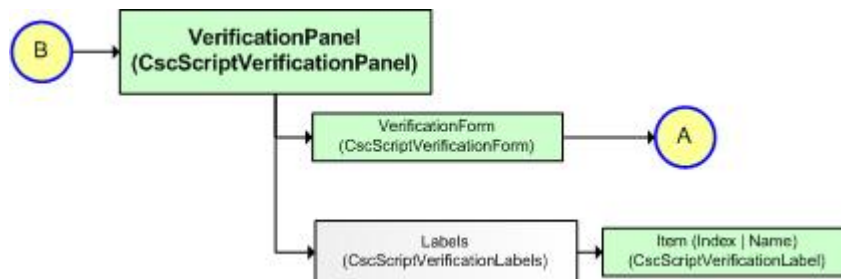
[Object Model of VerificationPanel \(Folder Data\)](#)

## Object Model of VerificationForm Object



Object Model VerificationForm (Document Data)

## Object Model of VerificationPanel Object



Object Model VerificationPanel (Folder Data)

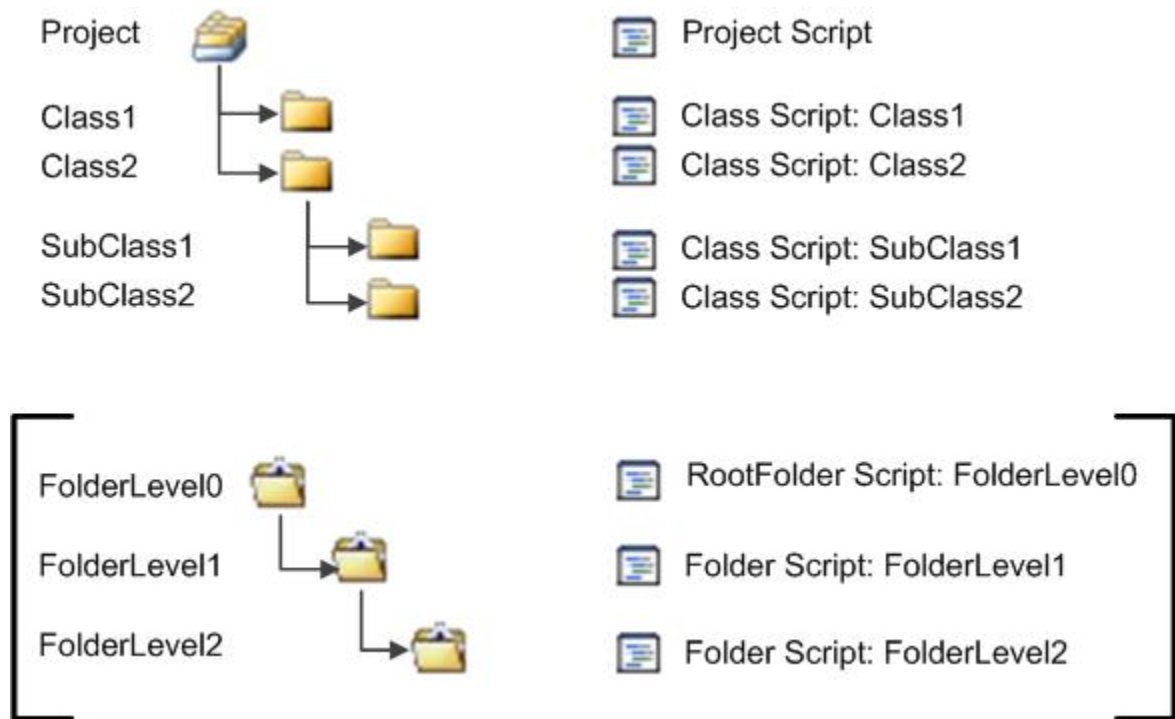
## Project

The Project (CscProject) object contains the complete definition of the processing workflow. During a running script this object can be accessed by 'Project.' and the corresponding property containing the desired information. A list of accessible functions and properties that are supported during scripting you find in the api reference (TODO, explain where you find it).

## Scripting Logic

The project structure is reflected in the scripting hierarchy.

In the following overview a project structure is shown and its reflection in the script sheets. The foldering related script sheets are only available if folders are enabled:



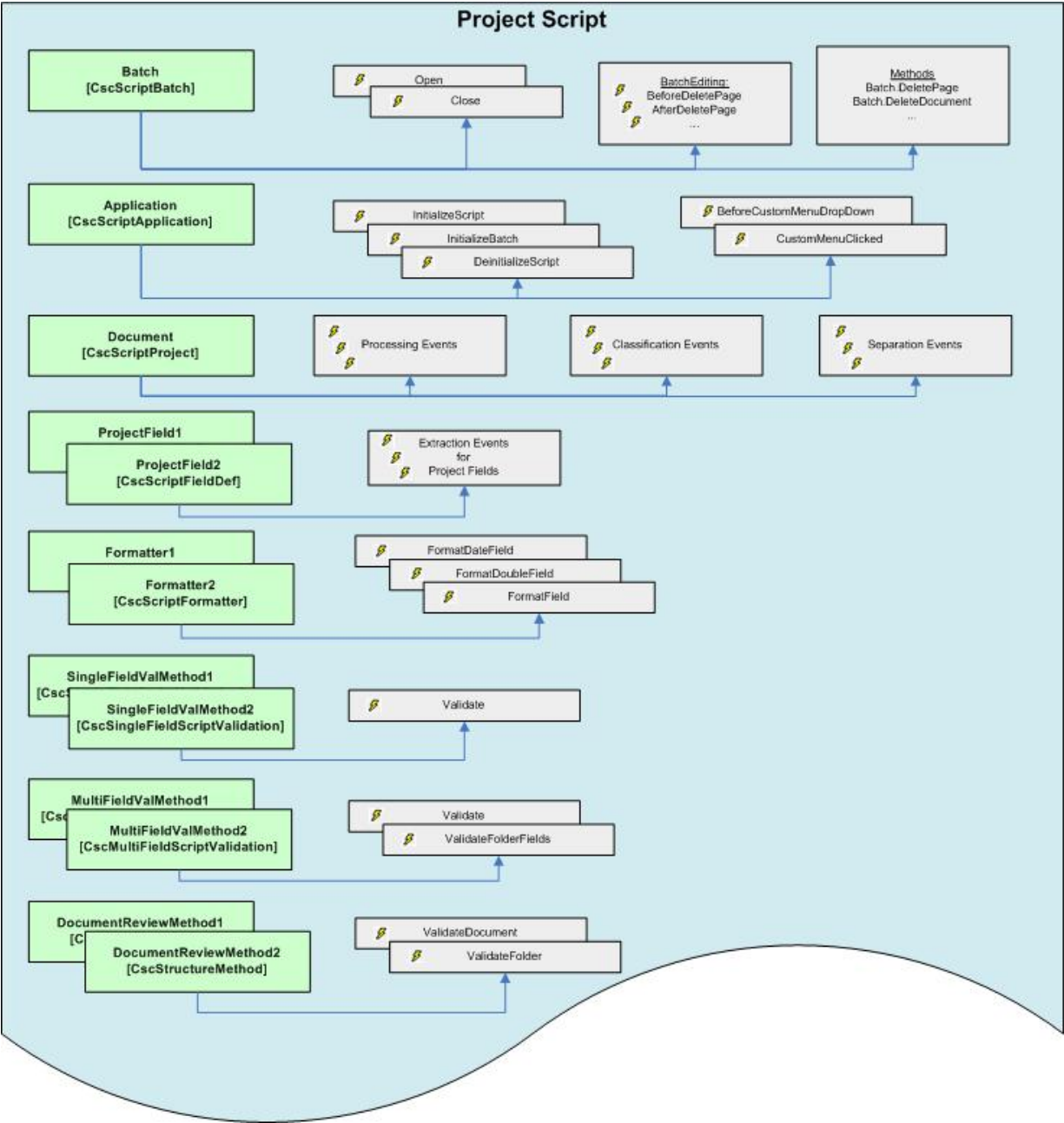
### Available Script Sheets

TODO: For the inheritance of the class structure and the folder structure please follow the link.

- [Project Script](#)
- [Class Script](#)
- [RootFolder Script](#)
- [Folder Script](#)

### Project Script

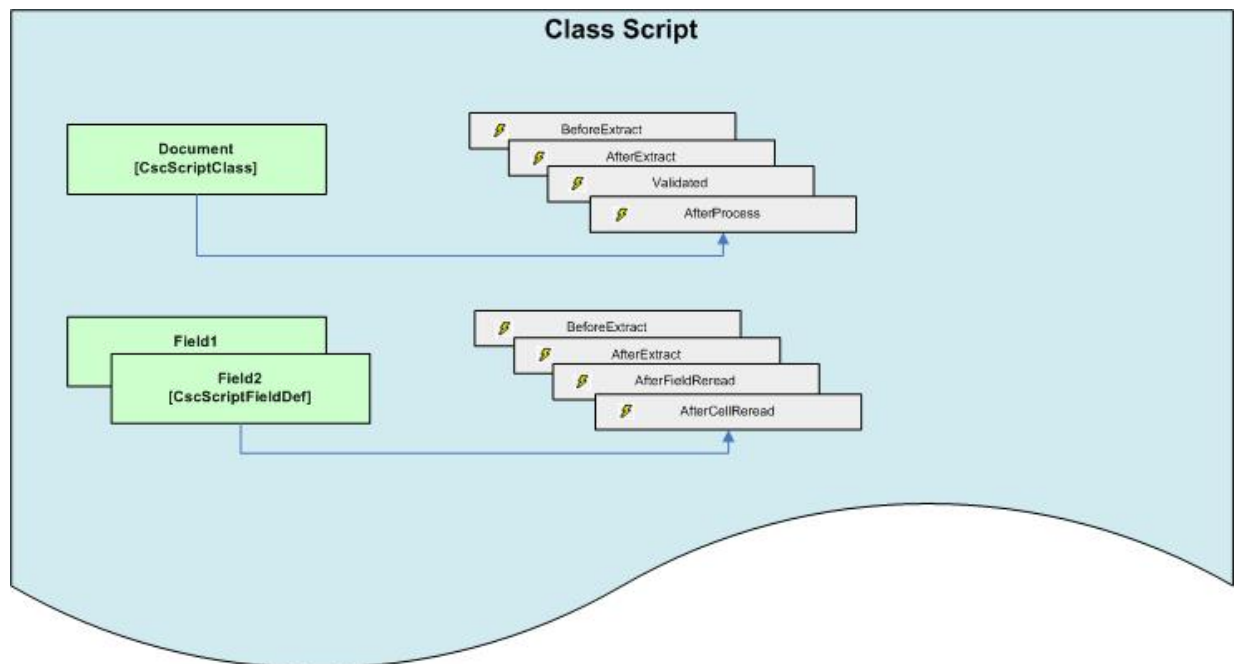
TODO



**Project Script**

**Class Script**

TODO

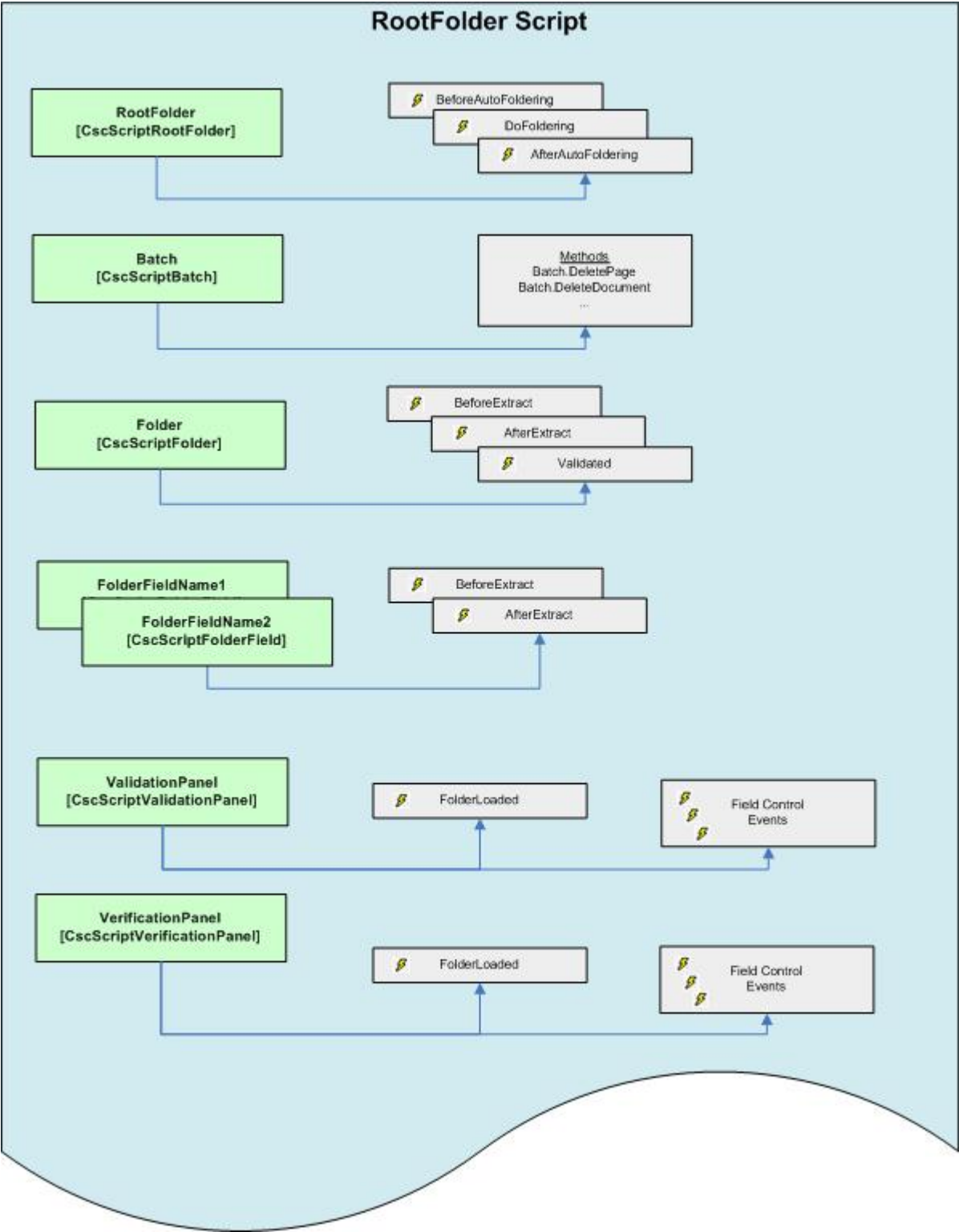


**Class Script**

**RootFolder Script**

TODO



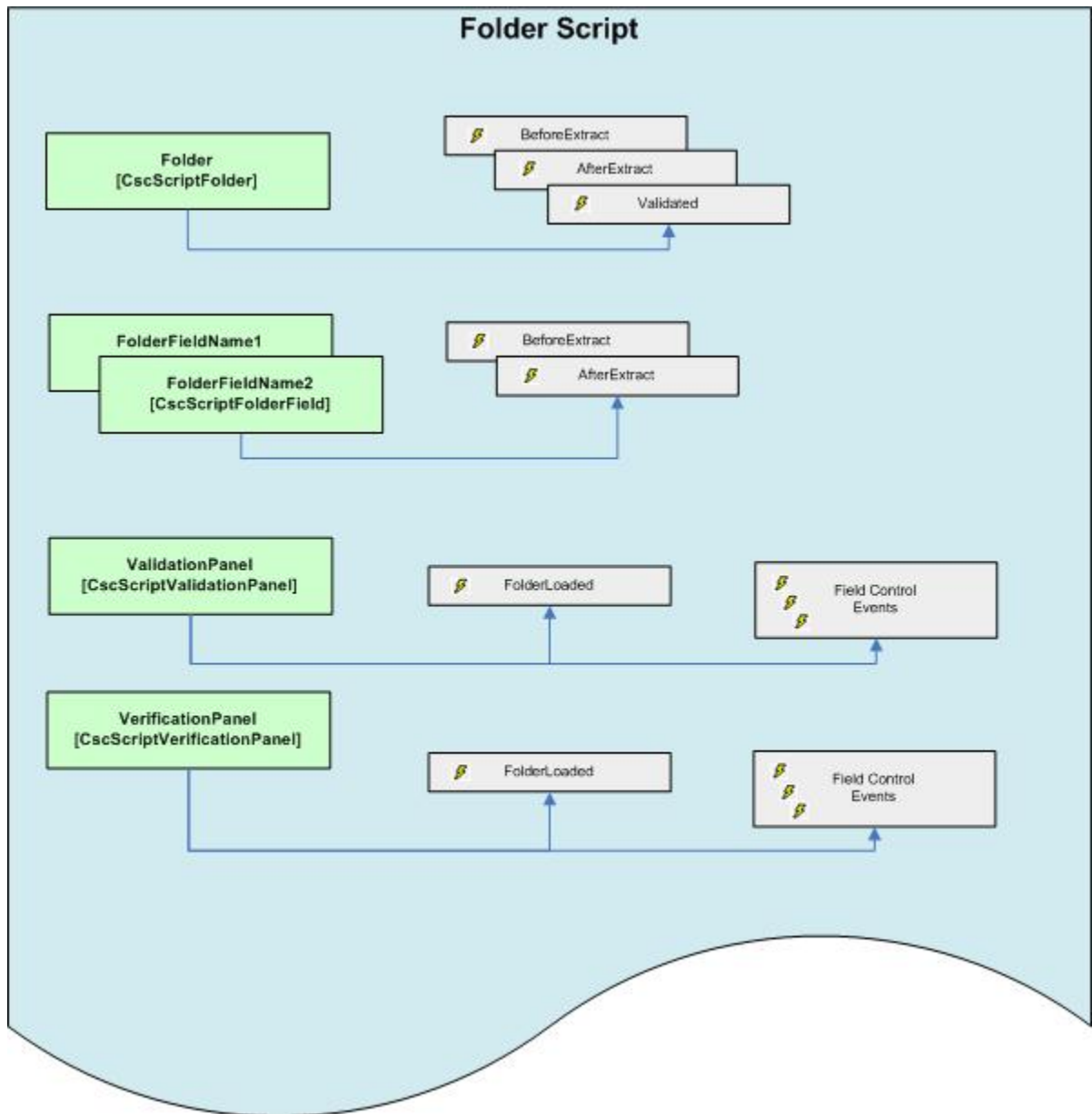


RootFolder Script

**Folder Script**

TODO





### Folder Script

## Inserting Script Events

This section describes step by step how to insert script code for document separation, classification, extraction, formatting, validation methods, folder and batch script event programming.

### ► To implement a document separation or classification script event

- 1 Open the Kofax Transformation - Project Builder.
- 2 Do one of the following:
  - From the View menu select Show Script.
  - Right-click a class, and select Show Script from the context menu.

- Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed.

- 3 Select the first sheet, and edit the project level for the classification.
- 4 From the Object drop-down list box, select Document and the event you want to implement (for example, `AfterClassifyImage`), and insert the code.

A procedure (for example, `Document_AfterClassifyImage`) is inserted into the edit area.

► **To implement an extraction script event (available for classes, for folders and on project level)**

- 1 Open the Kofax Transformation - Project Builder, and select the class or the folder for which you set up the script code..
- 2 Do one of the following:
  - From the View menu select Show Script.
  - Right-click a class, and select Show Script from the context menu.
  - Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed, as well as the sheet for the selected class or the selected folder containing the extraction events. The first sheet (project script) contains the classification events for the class on the project level and the extraction events of the project fields. The extraction events of the root folder are not in the project script sheet but in the script sheet of the root folder.

- 3 To implement the event that occurs at the beginning or end of the extraction, select Document from the Object drop-down list box and the corresponding `BeforeExtract` or `AfterExtract` event from the Proc drop-down list box. .
- 4 To implement a field extraction event, select a field from the Object drop-down list box and the corresponding `BeforeExtract` or `AfterExtract` event from the Proc drop-down list box.
- 5 To implement the event for an individual locator method, select a locator from the Object drop-down list box and `LocatorAlternatives` from the Proc drop-down list box. A procedure (for example, `MyField1_AfterExtract`) is inserted into the edit area where you insert the script code.

► **To implement a formatting script event**

- 1 Open the Kofax Transformation - Project Builder.
- 2 Add a script formatter (for example, "MyScriptFormatter") in the Project Settings window, and select the data type for it.
- 3 Copy the created Script Sample in the window to the clipboard.
- 4 Select the Project object in the Project Structure, and do one of the following:
  - From the View menu select Show Script.
  - Right-click a class, and select Show Script from the context menu.
  - Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed.

- 5 Select the first sheet to edit the project level (if it is not already displayed).
- 6 Do one of the following:

- Paste the copied sample code into the project script.
- Select the formatter object “MyScriptFormatter” that you have inserted from the Object drop-down list box and the event you want to implement (for example, select “FormatDateField” if you have selected “Date” as the field data type of the formatter). A procedure (for example, MyScriptFormatter\_FormatDateField) is inserted into the edit area where you insert the script code.

► **To implement a validation form script event**

- 1 Open the Kofax Transformation - Project Builder, and select the class for which you set up the script code.
- 2 Do one of the following:
  - From the View menu select Show Script.
  - Right-click a class, and select Show Script from the context menu.
  - Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed, as well as the sheet for the selected class containing the validation form events. The `ValidationForm` object is only available for a class if a validation form is defined.

- 3 To implement one of the events, select `ValidationForm` from the Object drop-down list box and the corresponding event from the Proc drop-down list box.

► **To implement a verification form script event**

- 1 Open the Kofax Transformation - Project Builder, and select the class for which you set up the script code.
- 2 Do one of the following:
  - From the View menu select Show Script.
  - Right-click a class, and select Show Script from the context menu.
  - Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed, as well as the sheet for the selected class containing the verification form events.

- 3 To implement one of the events, select `VerificationForm` from the Object drop-down list box and the corresponding event from the Proc drop-down list box.

► **To implement a structure method script event**

- 1 Open the Kofax Transformation - Project Builder.
- 2 On the Document Review tab of the Project Settings window, add a document review script method (for example, “MyDocReview”, “MyFolderReview”).
- 3 Select the type of document review script method. Set it to “Document Method” if the structure method is executed for a document or set it to “Batch Method” if the structure method validates the batch.
- 4 Copy the created Script Sample in the window to the clipboard.
- 5 Select the Project object in the Project Structure, and do one of the following:
  - From the View menu select Show Script.

- Right-click a class, and select Show Script from the context menu.
- Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed.

- 6 Select the first sheet to edit the project level (if it is not already displayed).
- 7 Do one of the following:
  - Paste the copied sample code into the project script.
  - Select the structure method object “MyDocReview” that you have inserted from the Object drop-down list box and the “ValidateDocument” event. A procedure (for example, `MyDocReview_ValidateDocument`) is inserted into the edit area where you insert the script code.

► **To implement a Batch script event for batch editing**

- 1 Open the Kofax Transformation - Project Builder.
- 2 On the General tab of the Project Settings window, activate “Foldering.”
- 3 Select the Project object in the Project Structure, and do one of the following:
  - From the View menu select Show Script.
  - Right-click a class, and select Show Script from the context menu.
  - Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed.

- 4 Select the first sheet to edit the project level (if it is not already displayed).
- 5 To implement the desired event, select “Batch” from the Object drop-down list box and the corresponding event from the Proc drop-down list box. A procedure (for example, `Batch_BeforeMoveDocument`) is inserted into the edit area where you insert the script code.

► **To implement a Folder script event for script based foldering**

- 1 Open the Kofax Transformation - Project Builder.
- 2 On the General tab of the Project Settings window, activate “Enable folders”
- 3 Select the Project object in the Project Structure, and do one of the following:
  - Select the root folder and select Show Script from the View menu .
  - Right-click on the root folder, and select Show Script from the context menu.
  - Select the root folder and click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed.

- 4 Select the first sheet to edit the root folder level (if it is not already displayed).
- 5 To implement the desired event, select “RootFolder” from the Object drop-down list box and the corresponding event from the Proc drop-down list box. A procedure (for example, `RootFolder_DoFoldering`) is inserted into the edit area where the script code will be inserted.

► **To implement a validation panel script event**

- 1 Open the Kofax Transformation - Project Builder, and select the folder for which you set up the script code.

**2** Do one of the following:

- From the View menu select Show Script.
- Right-click a folder, and select Show Script from the context menu.
- Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed, as well as the sheet for the selected class containing the validation form events. The `ValidationPanel` object is only available for a class if a validation panel is defined.

**3** To implement one of the events, select `ValidationPanel` from the Object drop-down list box and the corresponding event from the Proc drop-down list box.

► **To implement a verification panel script event**

**1** Open the Kofax Transformation - Project Builder, and select the folder for which you set up the script code.

**2** Do one of the following:

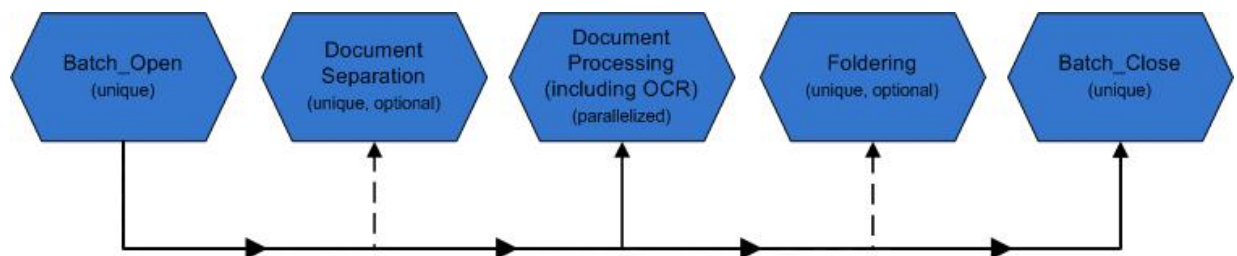
- From the View menu select Show Script.
- Right-click a folder, and select Show Script from the context menu.
- Click Show Script from the toolbar.

The WinWrap Basic Script Editor window is displayed, as well as the sheet for the selected class containing the verification panel events.

**3** To implement one of the events, select `VerificationPanel` from the Object drop-down list box and the corresponding event from the Proc drop-down list box.

# Server Events Scripting

During server processing for a single batch, the tasks are carried out according to the following figure:



## Batch processing tasks

- all events occurring in a single task are executed in a single process
  - transfer data using global variables between tasks; different tasks can be distributed across different processes
  - dependent on usage / configuration, some tasks are optional or distributed between the two server instances
- [Batch Processing](#) - The server module fires the standard [Application and Batch](#) events to enable a customized batch handling that are distributed differently due to the [parallelization](#) of processed batches and their sub tasks.
  - [Document Separation](#) - If activated, document separation is performed as a preparation step of the batch. There are two separation algorithms available. One is based on ... [TODO add here a link to the sub chapter](#) the other on ....[TODO add here a link to the sub chapter](#). The separation events are placed in the project script sheet.
  - [Document Processing](#) - the document processing can be divided into several sub steps:
    - [Classification](#) - Classification is the first step in document processing and can be treated independently from extraction although the extraction results can be used for classifying documents accordingly in a later step. All classification events are placed in the project script sheet.
    - [Extraction](#) - The main purpose of this processing step is to recognize and extract relevant items on a document. There are different types of extraction events for either the document, the fields, or a script locator. All events related to the extraction events are placed in the corresponding class script.
    - [Validation](#) and [validation methods](#) - The main purpose of the validation processing step is to prove the validity of the field content. This can be done either by built-in validation methods or customized scripted single or multiple field validation methods. The events of the scripted validation methods are placed in the project script sheet.
  - [Foldering](#) - If foldering is enabled, the foldering is executed after all documents are extracted and validated. The events of foldering are placed in script sheet of the root folder object.

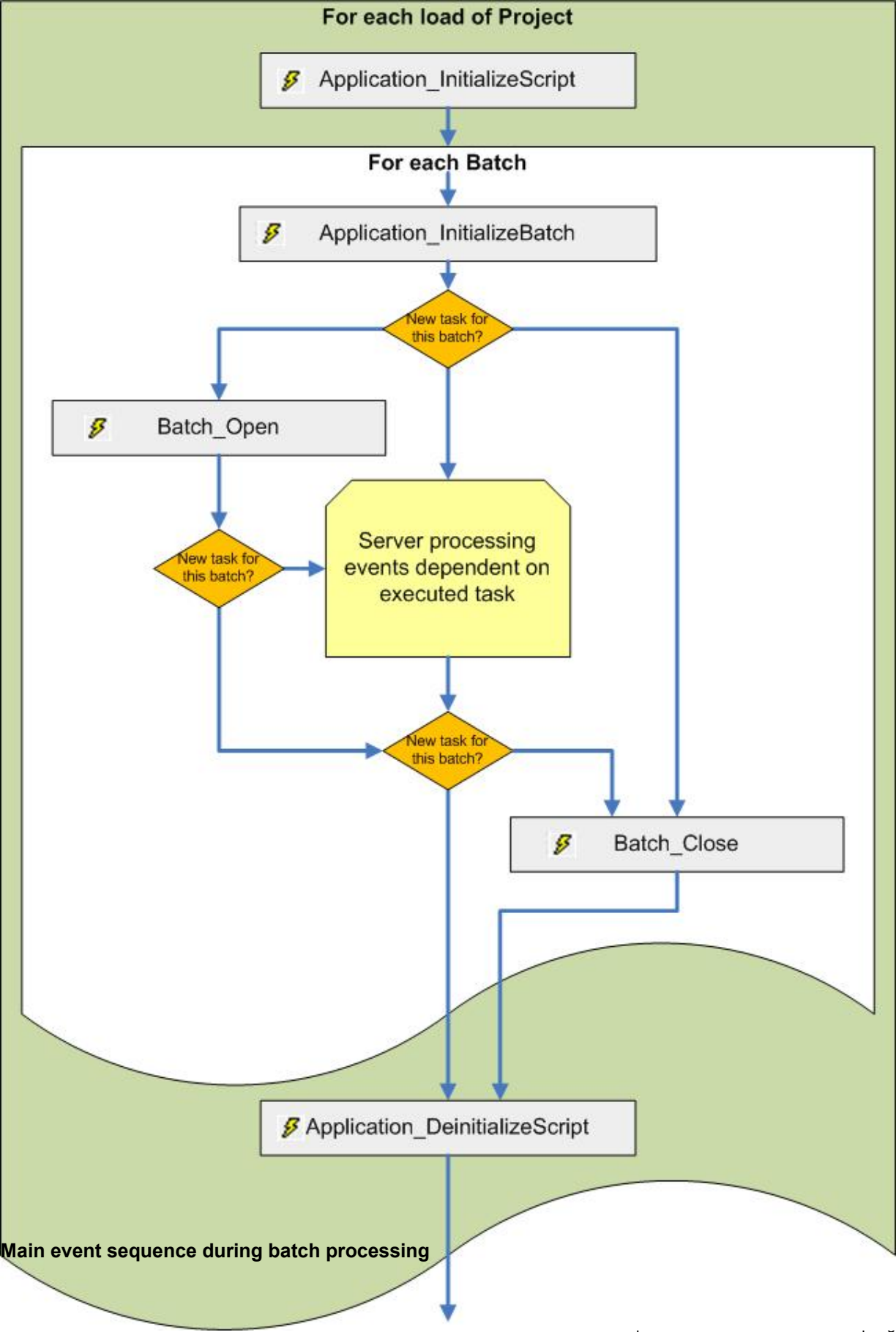
[Folder Extraction and Validation Events](#) - After executing the foldering events the extraction and validation events of the folders are executed. These events are placed in the corresponding script sheet of the folder object.

## Batch Processing

- tasks are distributed to the extraction processes via the server scheduler
  - when a (new) task is assigned a project is loaded, the scripting engine is initialized (Application\_InitializeScript) and the Application\_InitializeBatch fires
  - events are fired dependent on the task processing events
  - a single task fires the Batch\_Open event for each batch, and a single task (clarify?) fires the Batch\_Close event
  - the event Application\_InitializeBatch is fired when a task that must be executed for a different batch is skipped (passed?). Note that when this task contains (both?) a different batch and different project, Application\_DeinitializeScript finalizes the sequence (concluded/completed/what is the state?) and (whereby?) Application\_InitializeScript starts a new sequence
- each extraction process fires the events as shown in the figure below - or - the following figure illustrates the script events fired during batch processing
- when a task is passed that has to be executed for a different batch again the event Application\_InitializeBatch is fired, when this task contains not only a different batch but also a different project the sequence is finalized by firing the Application\_DeinitializeScript event and a new sequence is started with the Application\_InitializeScript for the new project.

The Open and Close event are also fired for all user interactive modules that support scripting. To distinguish between the opening module please use the property `Project.ScriptExecutionMode`.

The following figure illustrates the script events fired during processing batches:

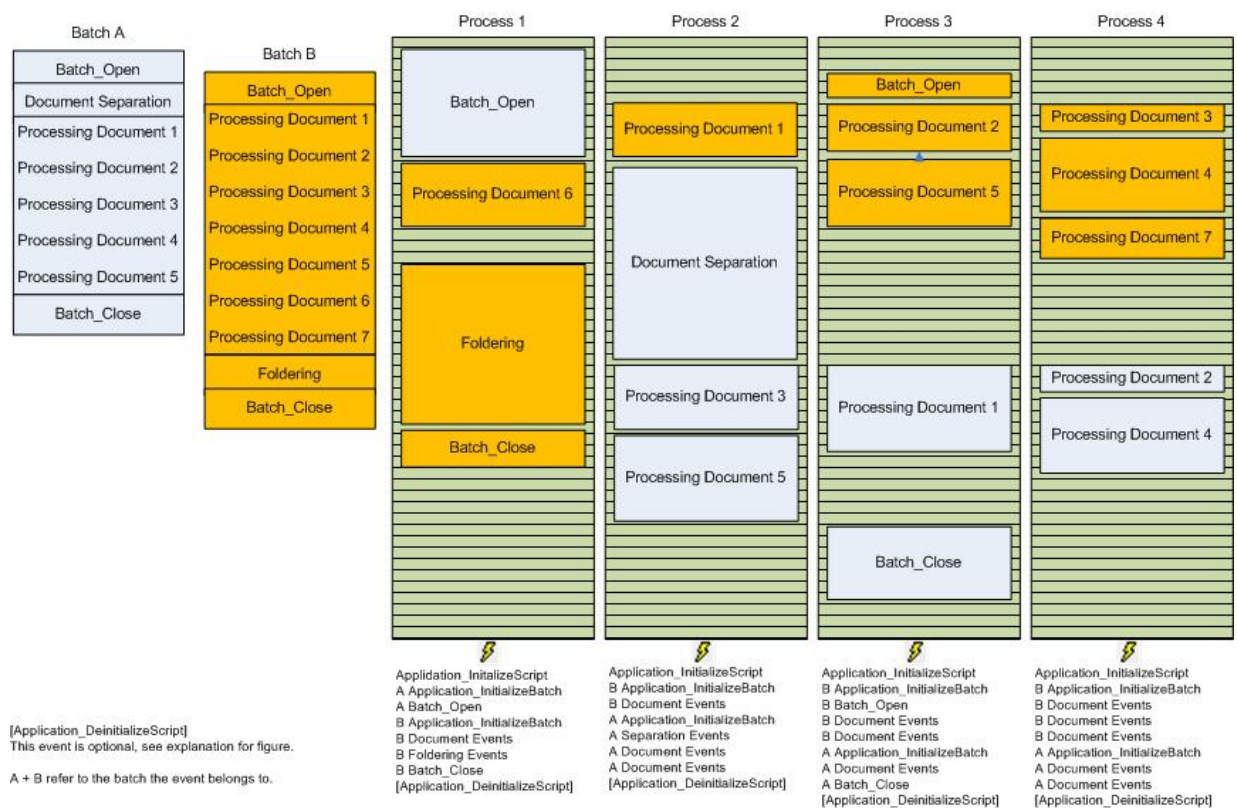




## Parallelization

Parallelization is...(performed/executed/enacted...?) when the following circumstances occur/ when the following status is met:

- 2 batches are opened within a short time of one another
- the tasks of these two batches are distributed across four working (?) processes
- explain which tasks can be parallelized, and the tasks that cannot
- explain the main statements of the figure
- explain which tasks can be parallized and which not
- explain the main statements of the figure



**Processing two batches with four working processes**

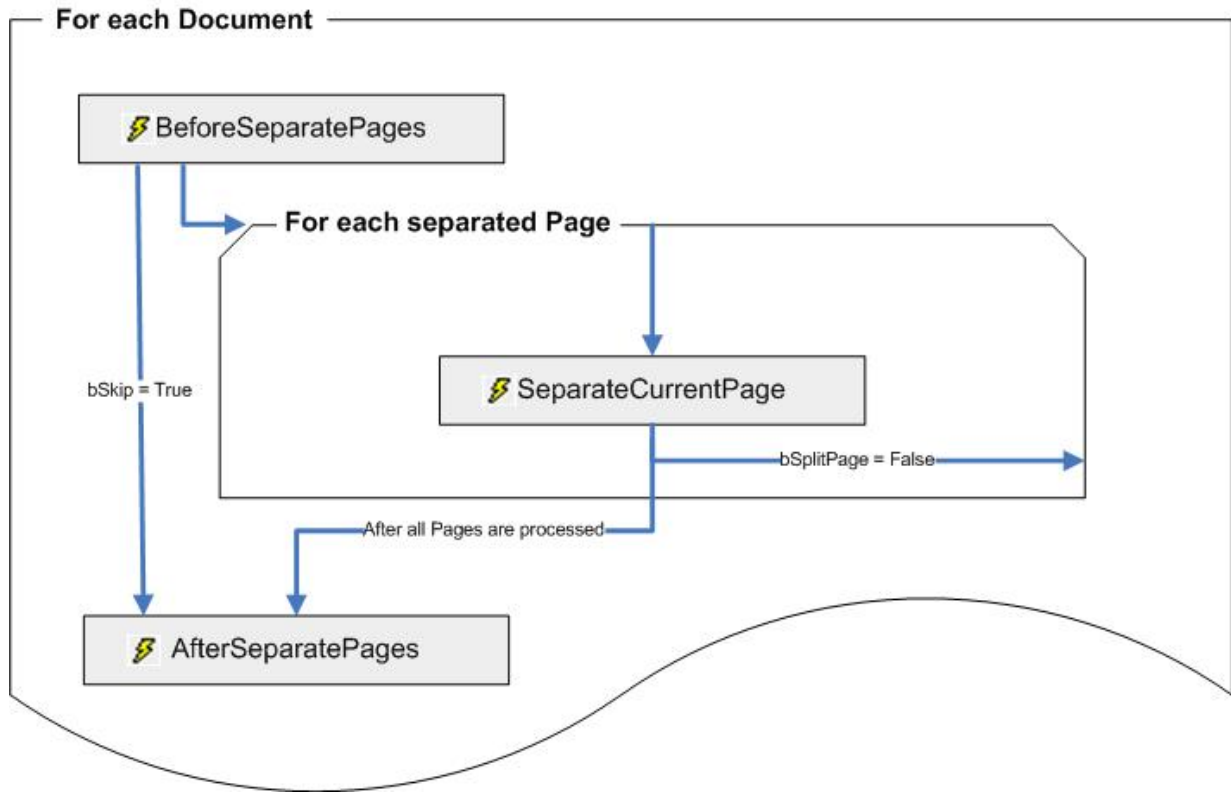
## Document Separation

TODO explain the difference of both algorithms.

## Standard Document Separation

Standard Document Separation events only occur during server processing.

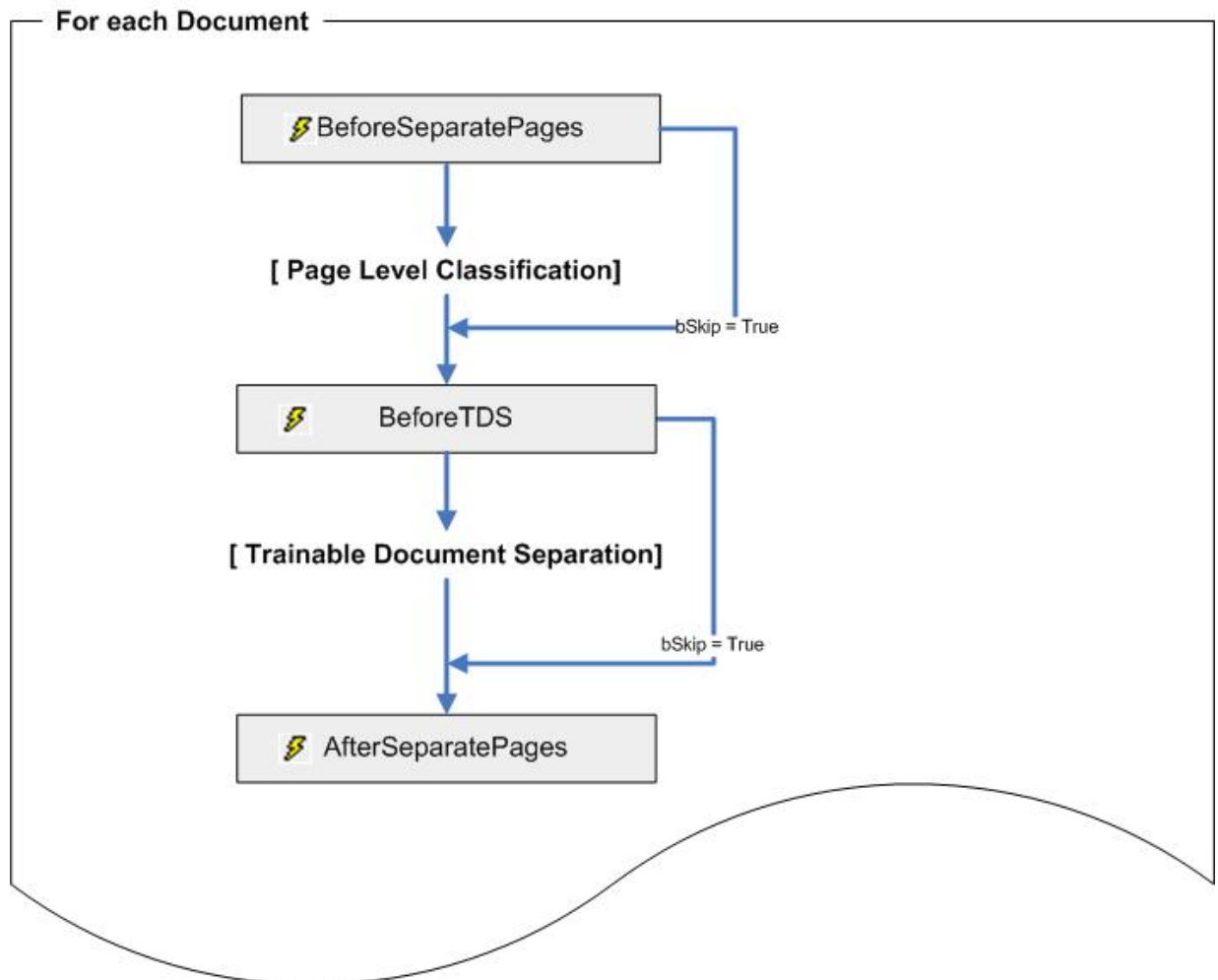
The general chronology of the document separation events is shown in the following figure:



### Trainable Document Separation (TDS)

Trainable Document Separation (TDS) events only occur during server processing.

The general chronology of the document separation events is shown in the following figure:



## Document Processing

The processing of the documents is mainly done in two steps. First the document is tried to be classified and then the settings of that specific class are applied to the document to retrieve the document data.

Each processing of a document is introduced by a `Document_BeforeProcessXDoc` in the project script sheet, because the incoming document is completely unknown. If project fields have been defined their extraction events are executed, then the classification events are following which are also part of the project script sheet.

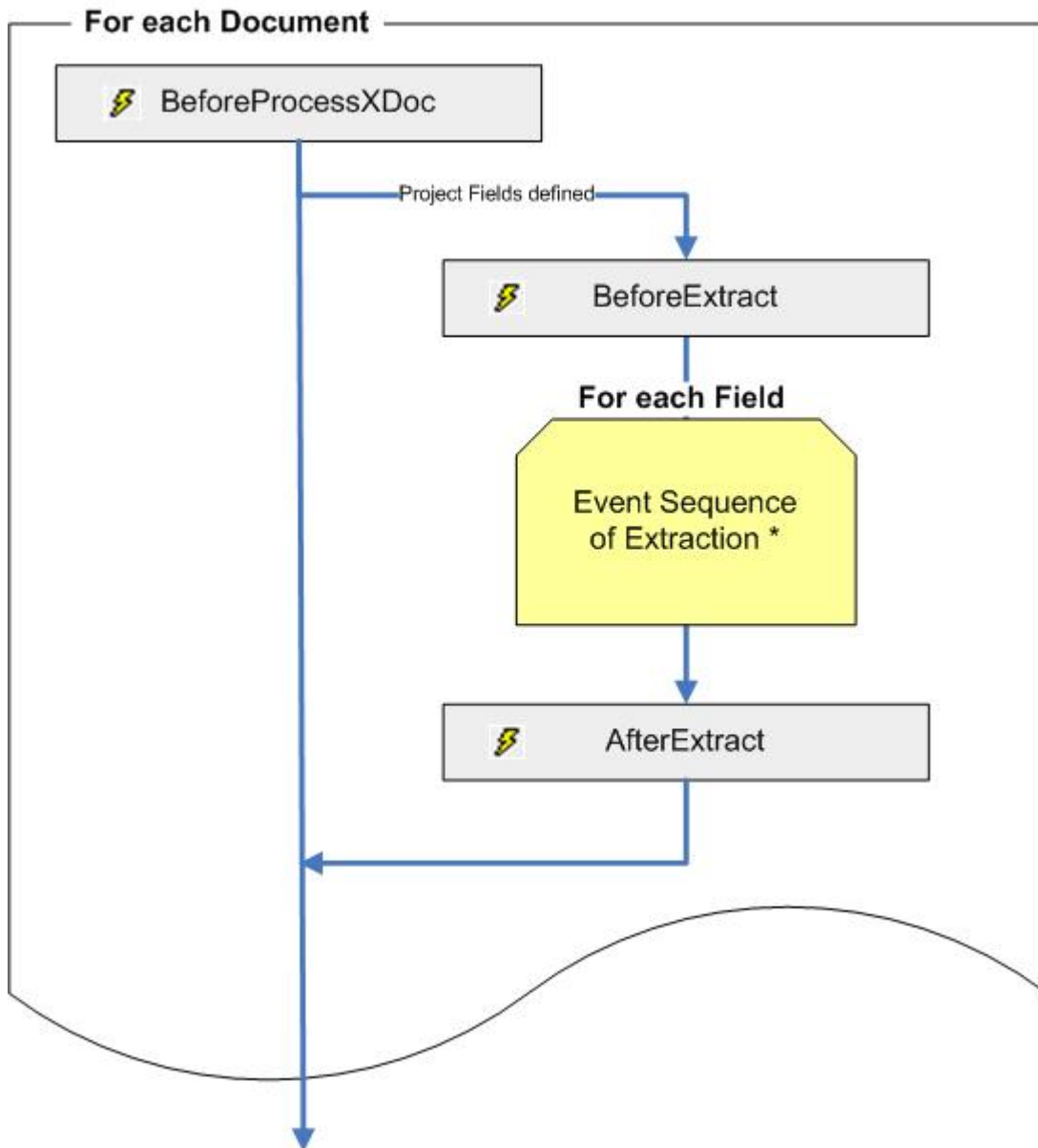
The extraction of the project fields can be used to classify the document by these extraction results, e.g. a bar code result is used to classify documents to a specific class.

Having a classified document the extraction of the document is performed. This means that all locator methods will be executed and by their assignment to the fields the document is getting its field results. The extraction comes along with extraction events that are following the defined class hierarchy, they are considering the field inheritance. When the extracted document belongs to a derived class, the extraction events for the inherited fields are also fired for all parent classes. This means that events are fired for the base class and then for the derived classes until the class is reached to that the document is classified to.

- project fields, usage, inheritance
- field inheritance

## Processing Project Fields

The script event sequence during document processing is illustrated in the following figure:



### Events before Classification and Extraction

\* For fields that are defined at the project level, the [Event Sequence During Extraction](#) is executed.

**Note** Note that the exception of the field inheritance not being considered and the Validated event not being executed.

For each document a **BeforeProcessXDoc** event is fired. This event can be used for suppressing OCR for several pages of a document, for this see also the included script example. If you have defined fields on project level the events **BeforeExtract** or **AfterExtract** are fired for each defined field on

project level. Between these two events the event sequence of extraction is executed with the difference to the normal field extraction being that no field inheritance and the `Validated` event is not available for project fields even though it is listed in the available event list for all project fields.

All project fields are set to `.Preserve = TRUE` after classification, so even if their definition is overwritten in a class the results are not overwritten. To enable the extraction to overwrite the result you have to set `.Preserve` to `FALSE` in the corresponding `Field_BeforeExtract` event of the class.

## Classification

The event chronology for classification events depends upon the project settings.

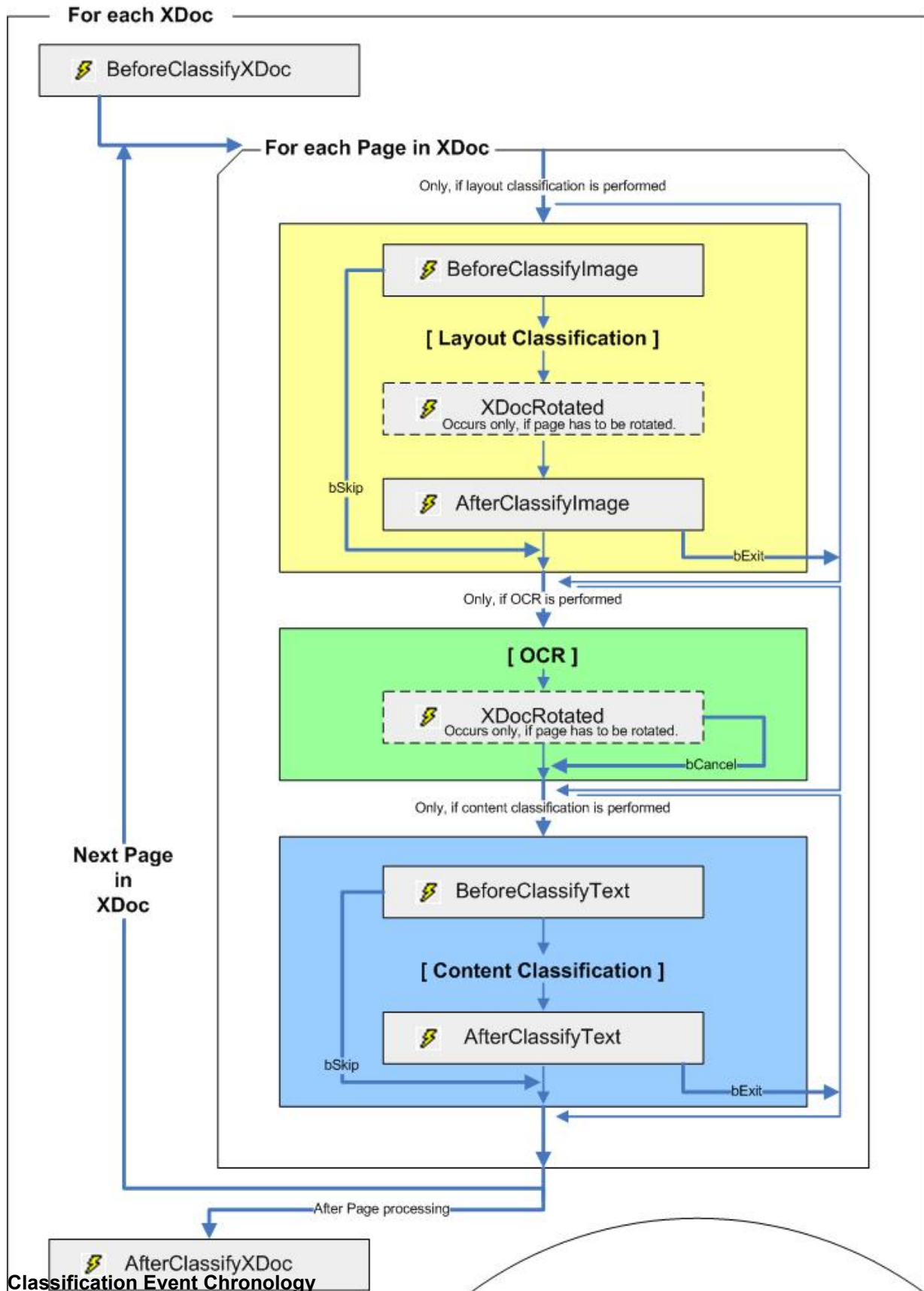
The events of the layout classification are either executed or not depending on whether or not the two classification algorithms are activated. This also applies when using the content classification. If the content classification is activated, OCR will be executed for the document and if it is not restricted to the first page OCR for the whole document is performed.

The `BeforeClassifyText/Image` and `AfterClassifyText/Image` can be limited to the first page or be executed for each page of the document depending on the classification settings.

The `XDocRotated` event occurs only when the Layout Classifier or the Content Classifier are active and are allowed to rotate a document page.

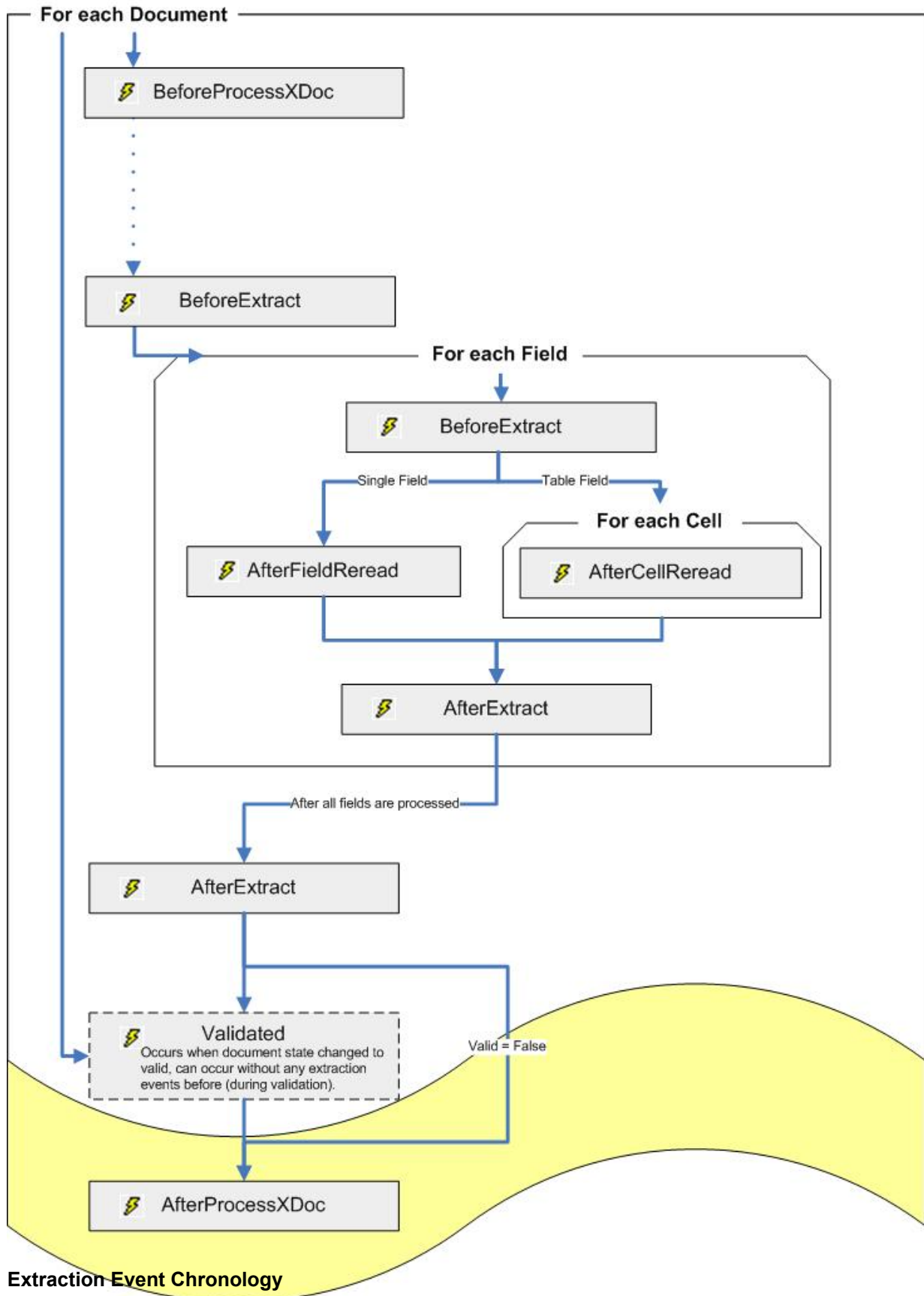
### Event Sequence During Classification

Classification events only occur during server processing. The following figure illustrates the script event sequence during classification:



## Extraction

Extraction events occur only during server processing; the `Validated` event can occur during server and validation process. The general chronology of extraction events is shown in the following figure.



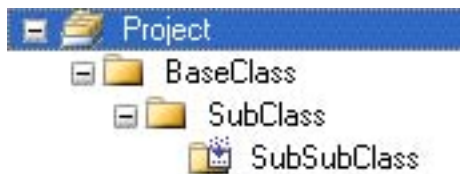


When the document in which fields are extracted belongs to a derived class, the Document\_BeforeExtract, Document\_AfterExtract, Field<n>\_BeforeExtract, and Field<n>\_BeforeExtract events for the inherited fields are also fired for all parent classes. This means that events are fired for the base class and then for the derived classes until the related class is reached.

### Extraction Events in Project Hierarchy

TODO rework this and move this to upper overview chapter or if it stays here make it a new section

If, for instance, you have a project with a class hierarchy as shown below, the following examples show the sequence of extraction events that are fired:



For the BaseClass, two fields (Field1 and Field2) are defined that are inherited by the derived classes. Field3 is defined on a SubSubClass level. The script locator method is defined for Field1 in the SubClass. When a document belonging to the SubSubClass class is extracted, these events occur in the following order:

```

BaseClass - Document_BeforeExtract
SubClass - Document_BeforeExtract
SubSubClass - Document_BeforeExtract
SubClass - MyScriptLocator1_LocateAlternatives
BaseClass - Field1_BeforeExtract
SubClass - Field1_BeforeExtract
SubSubClass - Field1_BeforeExtract
BaseClass - Field1_AfterFieldReread **
SubClass - Field1_AfterFieldReread **
SubSubClass - Field1_AfterFieldReread **
BaseClass - Field1_AfterExtract
SubClass - Field1_AfterExtract
SubSubClass - Field1_AfterExtract
BaseClass - Field2_BeforeExtract
SubClass - Field2_BeforeExtract
SubSubClass - Field2_BeforeExtract
BaseClass - Field2_AfterFieldReread **
SubClass - Field2_AfterFieldReread **
SubSubClass - Field2_AfterFieldReread **
BaseClass - Field2_AfterExtract
SubClass - Field2_AfterExtract
SubSubClass - Field2_AfterExtract
SubSubClass - Field3_BeforeExtract
SubSubClass - Field3_AfterFieldReread **
SubSubClass - Field3_AfterExtract
BaseClass - Document_AfterExtract
SubClass - Document_AfterExtract
SubSubClass - Document_AfterExtract
BaseClass - Document_Validated *
SubClass - Document_Validated *
SubSubClass - Document_Validated *
BaseClass - Document_AfterProcessXDoc
SubClass - Document_AfterProcessXDoc
SubSubClass - Document_AfterProcessXDoc
  
```

\* The validated events are only fired when the document is valid.

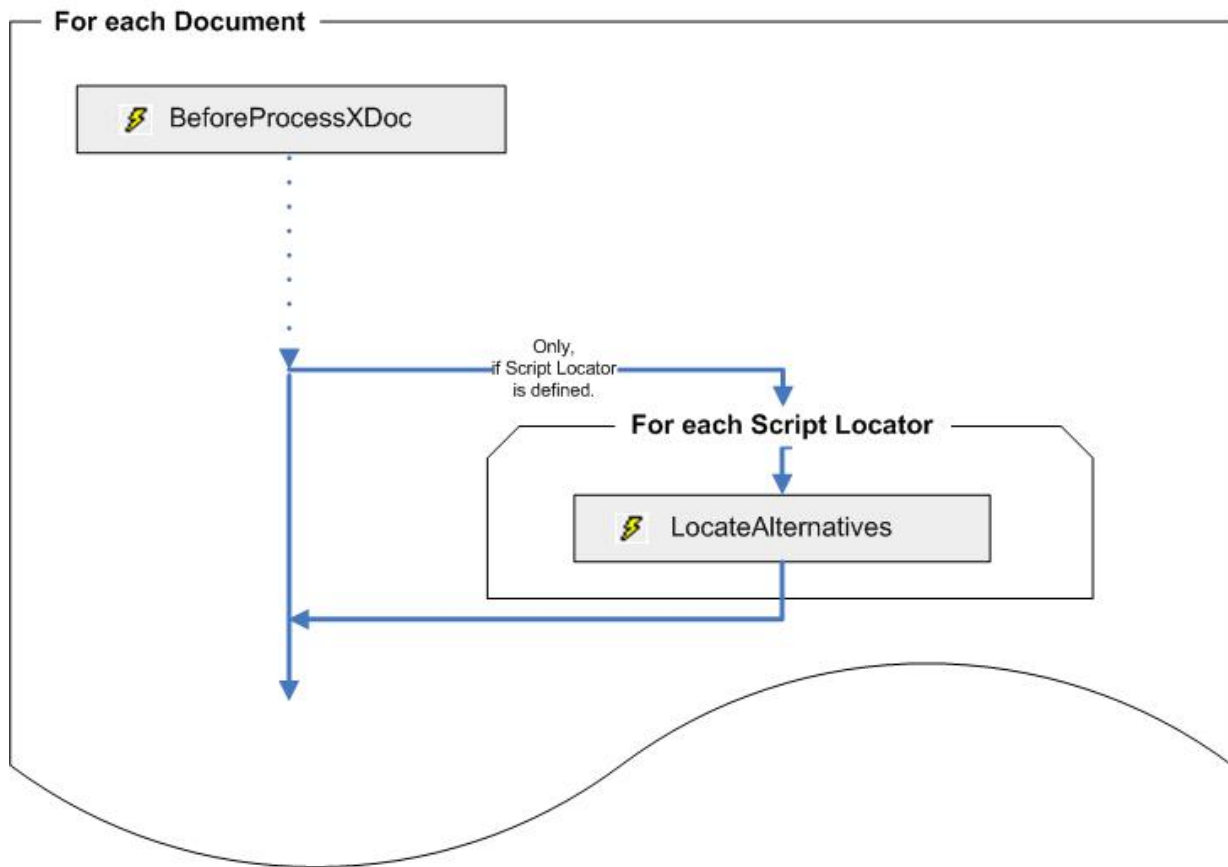
\*\* To get the reread events of a table cell or a field, you have to activate the reread option of a field or table column. This can be done from within the Field Properties window.

### Script Locator

You can define a custom locator using the script locator by means of the Sax Basic script engine. The script locator exits to a script event that either implements the location method in the VBA-compatible script language or calls a custom locator DLL.

The event is part of the class script sheet in that the locator is defined and follows the naming convention `ScriptLocatorName_LocateAlternatives`.

TODO example! with handling of alternatives



### Script Locator events

#### OCR Reread

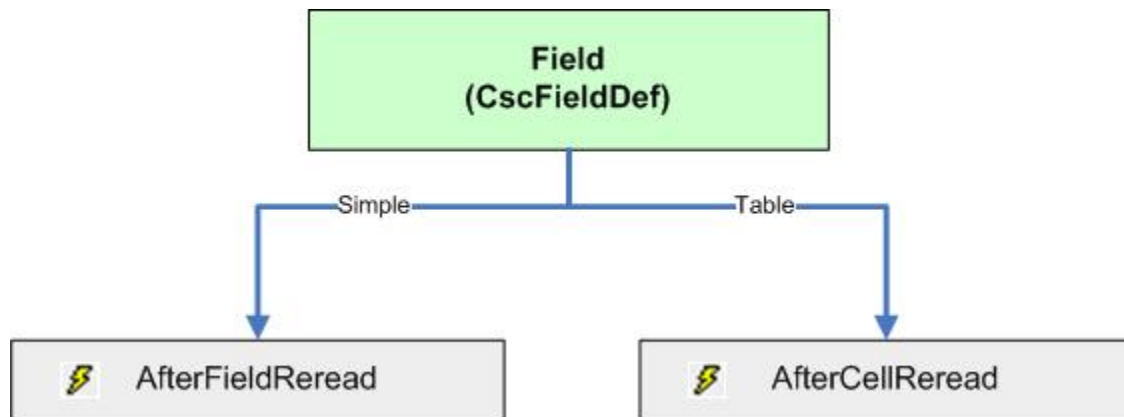
When a document is classified locally for the class, defined OCR settings can be applied per field. This enhances the OCR results because the document type is recognized. Typically this would be applied when rereading an area with handwriting recognition. The reread option of a simple or a table field is configured in its properties window and you can define a minimum confidence to accept the new OCR results as the new field result.

The Reread events occur only if an OCR Profile is specified in the Reread Options.

TODO explain more, when the event is coming ....

TODO add here an example for scripting

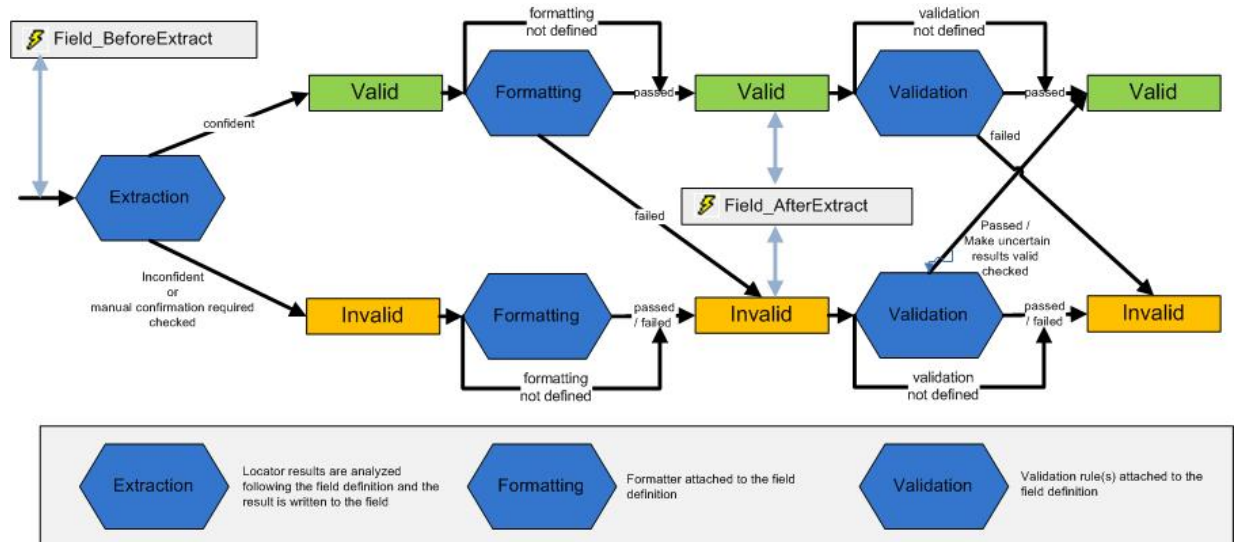
TODO explain the parameters



OCR reread events

## Validation

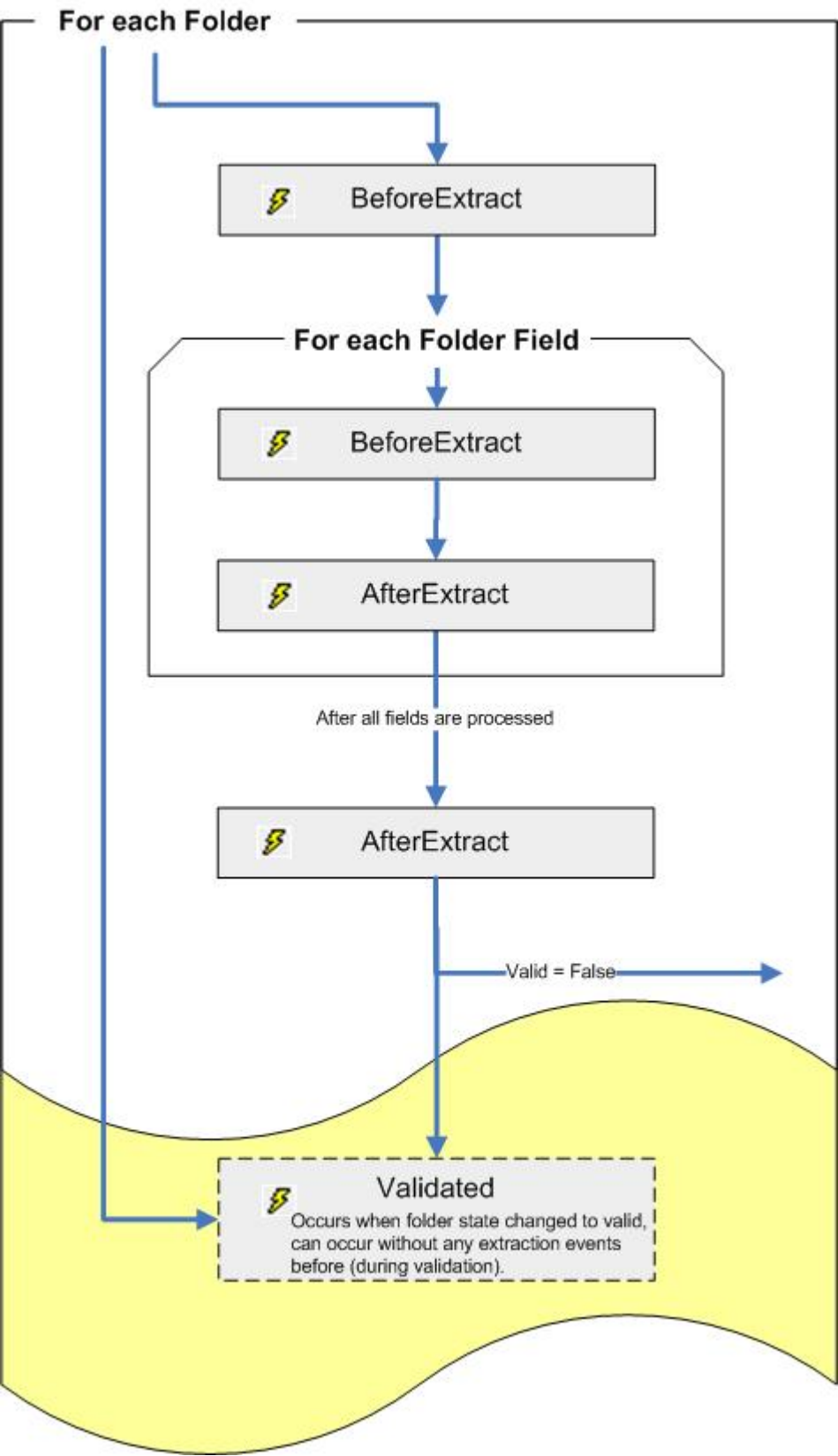
The field status (invalid/valid) is related to the results of the extraction (ExtractionConfident), the defined formatting and the applied validation rule that is attached to the field. During server processing state of the field is following the scheme in the flowchart below.



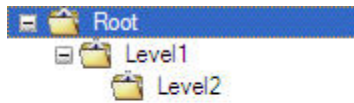
Field validation sequence in server

## Folder Extraction and Validation

Folder extraction is applied after the [folding](#) events, the corresponding folder extraction events occur only during server processing. The general chronology of extraction events of a folder is shown in the following figure.



The following example shows the sequence of extraction events that are fired when you have a project with the folder hierarchy shown below:



For each level two fields are defined as shown in the following:

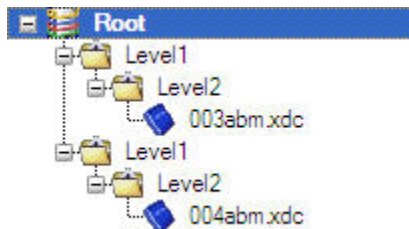
```

Root
  Root_Field1
  Root_Field2

Level1
  Level1_Field1
  Level1_Field2

Level2
  Level2_Field1
  Level2_Field2
  
```

A batch is processed having two documents:



The events then occur in the following order:

```

Level2 (containing 003.xdc)
  Folder_BeforeExtract
  Level2Field1_BeforeExtract
  Level2Field1_AfterExtract
  Level2Field2_BeforeExtract
  Level2Field2_AfterExtract
  Folder_AfterExtract
  Folder_Validated*

Level1 (containing folder Level2 with 003.xdc)
  Folder_BeforeExtract
  Level1Field1_BeforeExtract
  Level1Field1_AfterExtract
  Level1Field2_BeforeExtract
  Level1Field2_AfterExtract
  Folder_AfterExtract
  Folder_Validated*

Level2 (containing 004.xdc)
  Folder_BeforeExtract
  Level2Field1_BeforeExtract
  Level2Field1_AfterExtract
  Level2Field2_BeforeExtract
  Level2Field2_AfterExtract
  Folder_AfterExtract
  Folder_Validated*
  
```

```
Level1 (containing folder Level2 with 004.xdc)
    Folder_BeforeExtract
    Level1Field1_BeforeExtract
    Level1Field1_AfterExtract
    Level1Field2_BeforeExtract
    Level1Field2_AfterExtract
    Folder_AfterExtract
    Folder_Validated*
```

```
Root
    Folder_BeforeExtract
    RootField1_BeforeExtract
    RootField1_AfterExtract
    RootField2_BeforeExtract
    RootField2_AfterExtract
    Folder_AfterExtract
    Folder_Validated*
```

\* The Validated event occurs only if the folder gets valid after the extraction process.

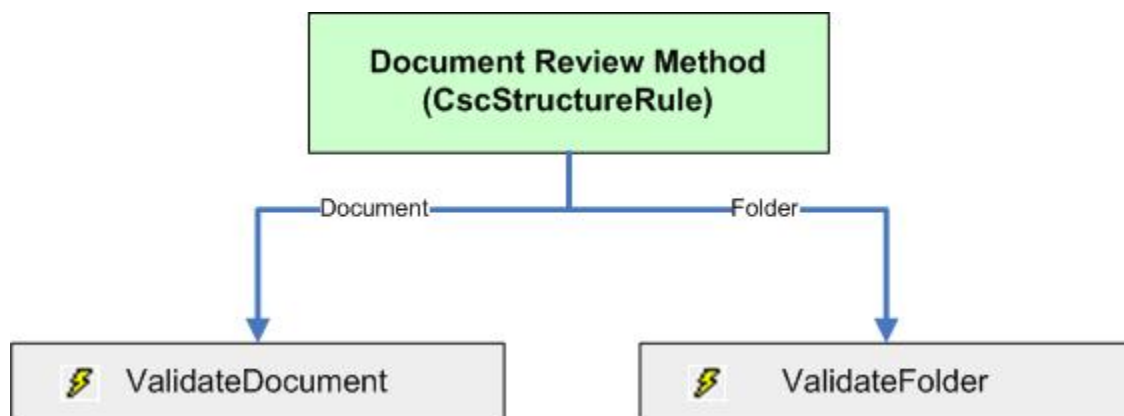
# Document Review Events Scripting

- [Application](#) - General events for all applications.
- [Batch](#) - Customized batch handling is enabled from the Document Review module via standard batch events.
- [Document Review Methods](#) - The `ValidateFolder` event of the `Batch` object and the `ValidateDocument` event of the `Document` object are fired when opening a batch in the Document Review module.
- [Batch Editing](#) - During document review batch editing events are initiated by the user and attached to the batch object to be placed on the project script sheet.
- [Customized Menus](#) For Document Review customized menus can be defined and implemented via script.

## Document Review Methods

Document Review methods are added and created for Project Settings. From Project menu select Project Settings to open the window. During the creation of the methods, sample script code is provided that can be copied directly into the project script.

The Document Review events occur only during the document review process. They are executed for every change in the batch. The events of the Document Review methods are part of the project script sheet. For a Document Review method depending on the method type that is defined during creation of the Document Review method the following events occur:



Event of Document Review Method for Folder

## Batch Editing Events

The batch editing events are initiated by the user during Document Review. \*Some of these events are related to foldering and are fired only if foldering is activated and if user action is allowed during Document Review. The events are attached to the [Batch](#) object and are inserted into the project script sheet. For the following user actions one or a sequence of events are fired:

- [Deleting Document](#)
- [Deleting Page](#)
- [Merging Document](#)
- [Moving Page](#)
- [Moving Document](#)
- [Rotating Page](#)
- [Splitting Document](#)
- [Confirming Class](#)
- [Changing Class](#)
- [Overriding / Restoring Document Problem](#)
- [Overriding / Restoring Folder Problem](#)

\*Batch editing is not available if folders are enabled.

## Events for Document Review's Customized Menu Commands

The following Document Review events occur during the document review process if a customized menu is defined.

- `Application_BeforeCustomMenuDropDown` - Use this event to activate or deactivate a customized menu command.
- `Application_CustomMenuClicked` - Use this event to define the action that occurs when the operator selects the menu command.

### Example. Opening an Image in an External Viewer

Selecting the “Open Image” menu item opens an image in an external viewer.

```
Private Sub Application_CustomMenuClicked(ByVal MenuName As String, ByRef pXFolder As
CASCADELib.CscXFolder, ByVal DocIndex As Long, ByVal PageIndex As Long, ByVal bDocSelected As
Boolean)
    'if "Open Image" menu item is selected then open the selected image in external viewer
    If MenuName = "Open Image" Then
        If (PageIndex > -1) And (DocIndex > -1) Then
            OpenImage(pXFolder.DocInfos(DocIndex).ImageFilename(PageIndex))
        End If
    End If
End Sub

'This function will open the file provided in the imgpath parameter with associated viewer
Function OpenImage(ByVal imgpath As String)
    Dim objShell
    objShell = CreateObject("Shell.Application")

    objShell.ShellExecute(imgpath, "", "", "open", 1)

    objShell = Nothing
End Function
```

### Example. Disabling the Open Image Command

```
Private Sub Application_BeforeCustomMenuDropDown(ByVal MenuName As String, ByRef pXFolder As
CASCADELib.CscXFolder, ByVal DocIndex As Long, ByVal PageIndex As Long, ByRef bDisableMenuItem
As Boolean)
    'if no document or page is selected then disable "Open Image" menu item
    If MenuName = "Open Image" Then
        If (DocIndex < 0 Or PageIndex < 0) Then
            bDisableMenuItem = True
        End If
    End If
End Sub
```



```
End If  
End Sub
```

# Validation Module Events Scripting

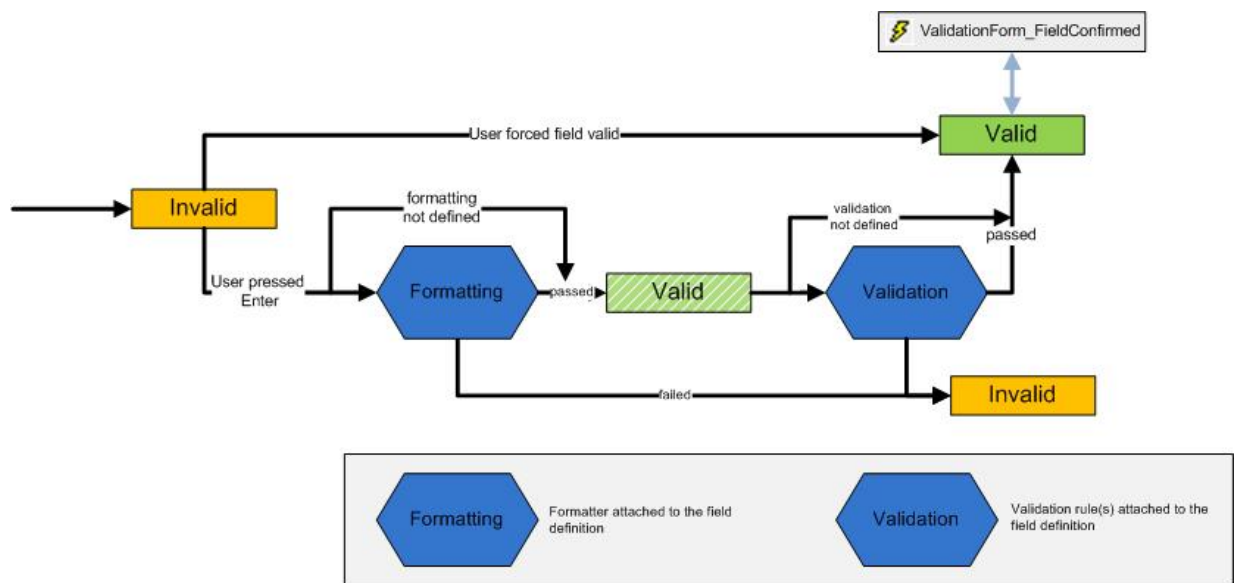
- [Application](#) - General events for all applications.
- [Batch](#) - The validation module fires the standard batch events to enable customized batch handling.
- [User interactive validation](#) and [validation methods](#) - The validation methods are executed on project level and are available for defined single and multi-field validation methods in combination with formatters the validity of a field is controlled when it is changed by a user.
- [Validation Layout Events](#) - If a validation layout is defined for a class a [ValidationForm](#) object is available in the corresponding class script. Several events are available for customization.

If a validation layout is defined for a folder, an object is available in the folder script sheet called [ValidationPanel](#) that provides several events for customization.

- [Batch Editing](#) - The batch editing events are initiated by user actions during document review and are attached to the batch object and placed on the project script sheet.

## Validating Fields

TODO



Process of user interactive validation

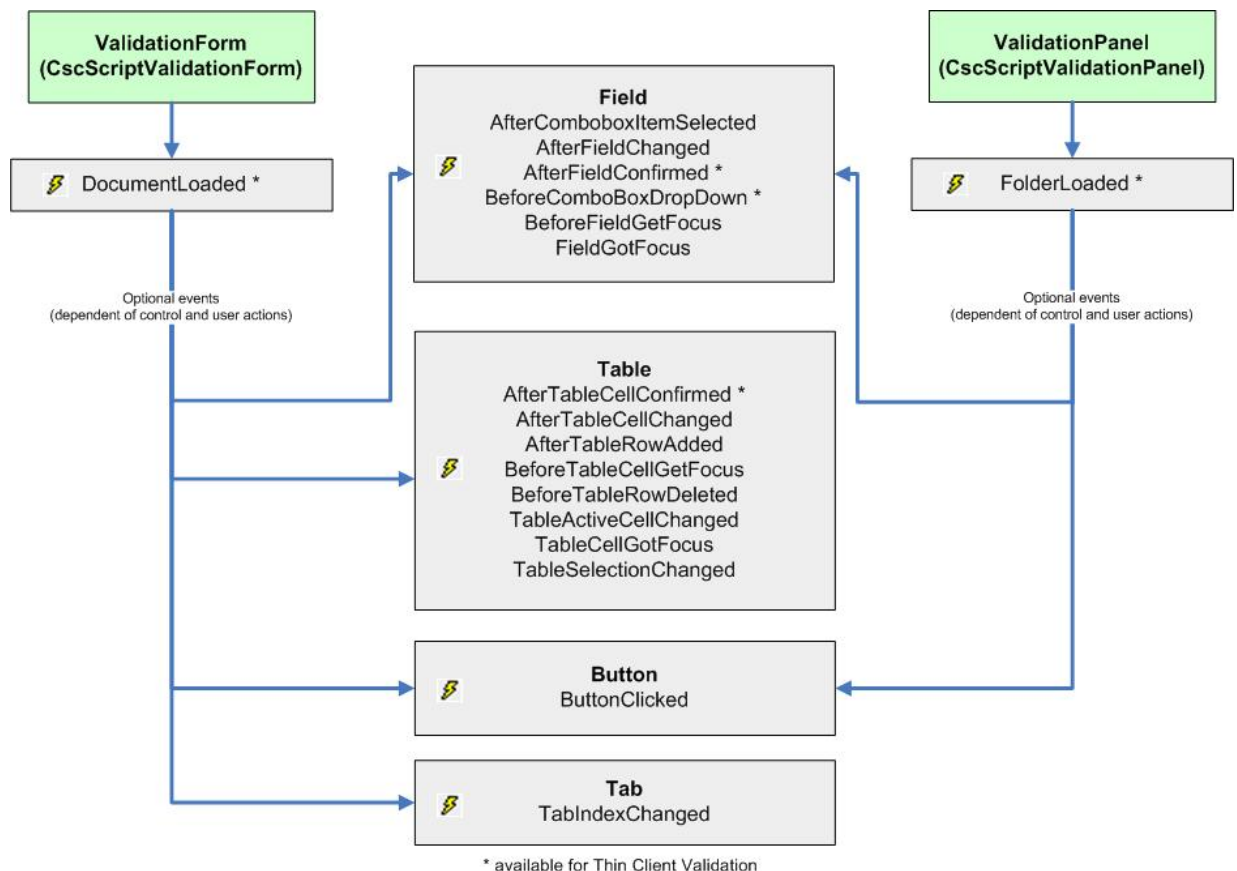
## Validation Layout Events

The [ValidationForm](#) object represents the document area. The events of the ValidationForm object are defined in the corresponding class script. It is only available if a validation layout is defined for

the class. The DocumentLoaded event is fired for each document loaded. All other events are fired depending on defined controls and user action on the ValidationForm object.

The events of the [ValidationPanel](#) object are defined in the corresponding folder script. It is available only if a validation layout is defined for the folder (the default layout is not accessible via script). The FolderLoaded event is fired for each folder loaded. All other events are fired depending on defined controls and user action on the ValidationPanel object.

The Thin Client Validation module does not support as many script events as rich-client Validation. The supported script events are flagged in the picture below.



### Events of the ValidationForm and the ValidationPanel Object grouped by control type

[Events of the Field Control](#)

[Events of the Table Control](#)

[Events of the Button Control](#)

[Events of the Tab Control](#)

### Events of the Field Control

AfterComboBoxItemSelected  
 AfterFieldConfirmed \*  
 BeforeComboBoxDropDown \*  
 BeforeFieldGetFocus  
 FieldGotFocus

## Events of the Table Control

AfterTableCellConfirmed \*  
 AfterTableCellChanged  
 AfterTableRowAdded  
 BeforeTableCellGetFocus  
 BeforeTableRowDeleted  
 TableActiveCellChanged  
 TableCellGotFocus  
 TableSelectionChanged

## Events of the Tab Control

TabIndexChanged

## Events of the Button Control

ButtonClicked

## Batch Editing Events

The batch editing events are initiated by user actions during validation. Some of these events are related to foldering and are only fired if foldering is activated and if the user action is allowed during validation. The events are attached to the [Batch](#) object that is available in the project script sheet. For the following user actions one or a sequence of events are fired:

- [Adding Document](#)
- [Adding Page](#)
- [Copying Document](#)
- [Creating Document](#)
- [Creating Folder](#) \*
- [Deleting Document](#)
- [Deleting Folder](#) \*
- [Deleting Page](#)
- [Merging Document](#)
- [Merging Folder](#) \*
- [Moving Document](#)
- [Moving Folder](#) \*
- [Rotating Page](#)
- [Splitting Document](#)
- [Splitting Folder](#) \*

\* occurs only if folders are enabled

# Verification Module Events Scripting

- [Application](#) - General events for all applications.
- [Batch](#) - The verification module fires the standard batch events to enable a customized batch handling.
- [Verification Layout Events](#) - For accessing the default layout of a [VerificationForm](#) an object is available in the corresponding class script that provides several events for customization (only available if fields are set to be verified).

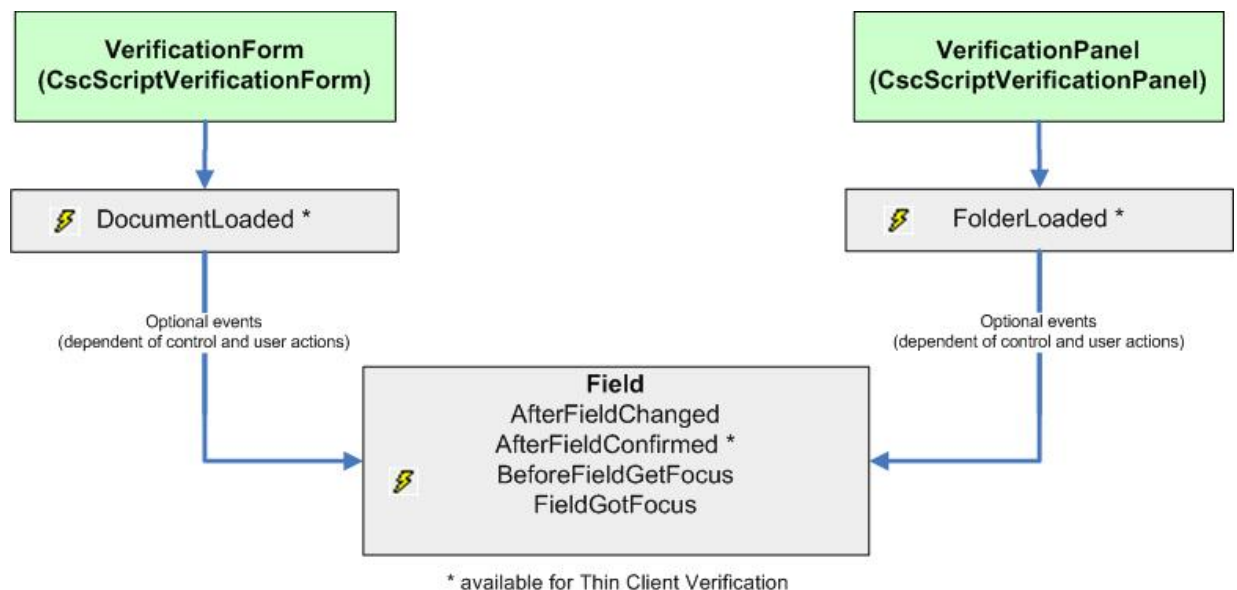
For the default layout of a folder an object is available in the folder script sheet called [VerificationPanel](#) that provides several events for customization (only available if fields are set to be verified).

## Verification Layout Events

The [VerificationForm](#) object represents the document area. The events of the VerificationForm object are defined in the corresponding class script. It is only available if at least one field is set to be verified for the class. The DocumentLoaded event is fired for each document loaded. All other events are fired depending on user action on the VerificationForm object.

The [VerificationPanel](#) object represents the folder area. The events of the VerificationPanel object are defined in the corresponding folder script. It is only available if at least one field is set to be verified for the folder. All other events are fired depending on user action on the VerificationPanel object. The Thin Client Verification module does not support as many script events as rich-client Verification. The supported script events are flagged in the picture below.

TODO remove thin clients comments and asterisks from graphic.



Events of the VerificationForm and the VerificationPanel object grouped by control type

# Applying Scripting

This topic contains a description of all scripting possibilities that are supported across several modules. Module specific restrictions are mentioned explicitly.

TODO add more general overview here about the included chapters.

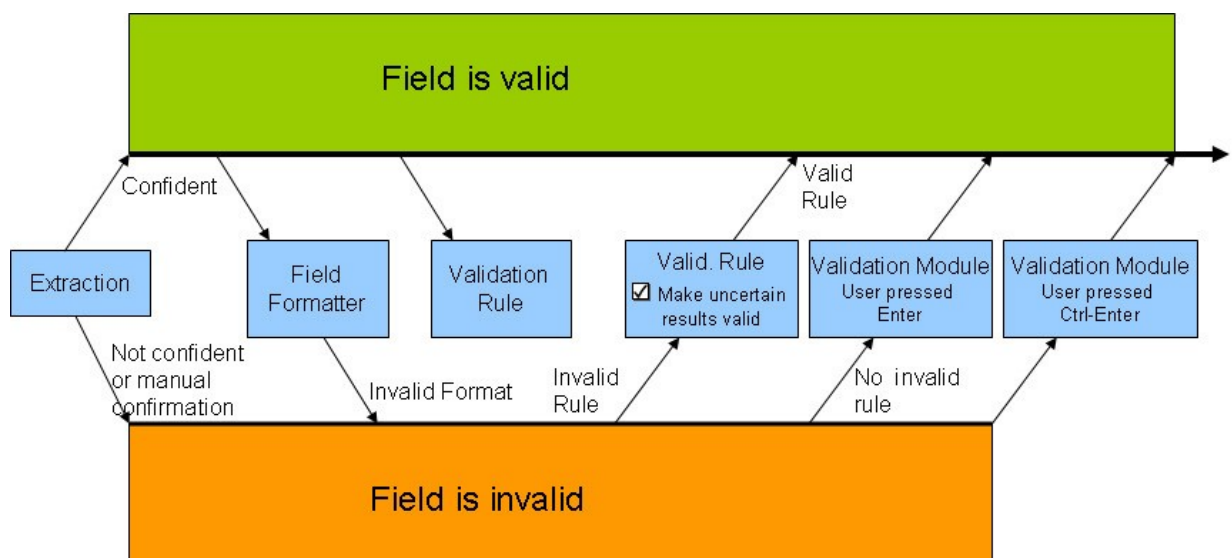
## Formatting

TODO.

## Validating

The following figure shows the validation process sequence, which mimics the order in which the validation checks occur. After extraction, the field formatter and then the validation rules are processed for each field as defined. The arrows in the figure show how the field state may change during validation checks. For example, a field formatter can either keep the current state of the field or change a “valid” field state to “invalid”, but it cannot change an “invalid” field state to “valid”. A validation rule can only change a field's state from “invalid” to “valid”, when “Make uncertain results valid” is selected in the properties of the validation rule.

**Important** The field's property `CscXDocField.Valid` can only be used in script to return the field's status, but not to set the status.



**Validation sequence and possible field state changes**

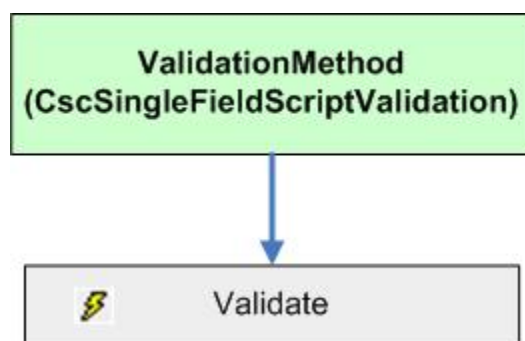
## Server Module Validation Events

Validation methods are added and created for Project Settings. To open the window, select Project Settings from the main menu. During the creation of the validation methods, sample script code is provided that can be copied directly into the project script.

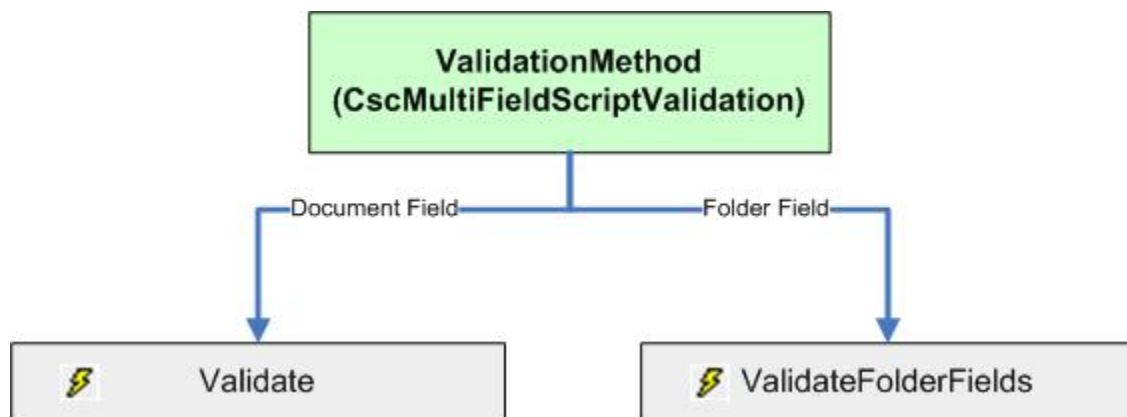
### Event for Validation Methods

The validation events occur during the server and validation processes. Remember when using message boxes, they may stop the complete server processing. You can use the property `Project.ScriptExecutionMode` to distinguish between server and validation processing and if multiple instances are configured for server or validation the running instance can be retrieved by the property `Project.ScriptExecutionInstance`.

For a single field and a multi-field script validation method the following events occur:



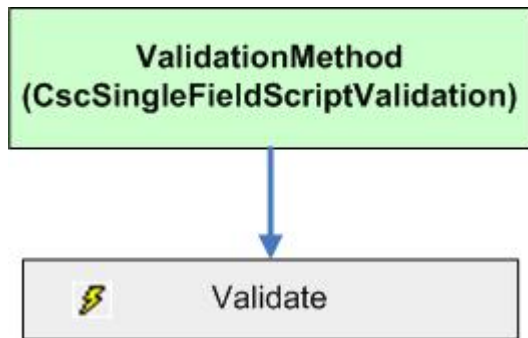
Event of Single Field Script Validation Method



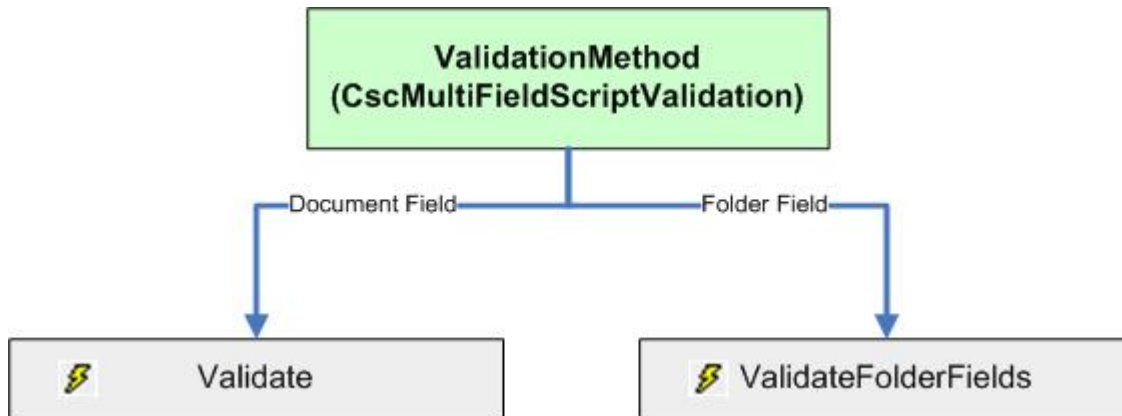
Event of Multi-Field Script Validation Method

## ValidationMethod Events

The validation events occur during the Server and validation process. Remember when using message boxes, they may stop the complete Server processing. You can use the property `Project.ScriptExecutionMode` to distinguish between server and validation processing. For a single field and a multi-field script validation method the following events occur:



Event of Single Field Script Validation Method

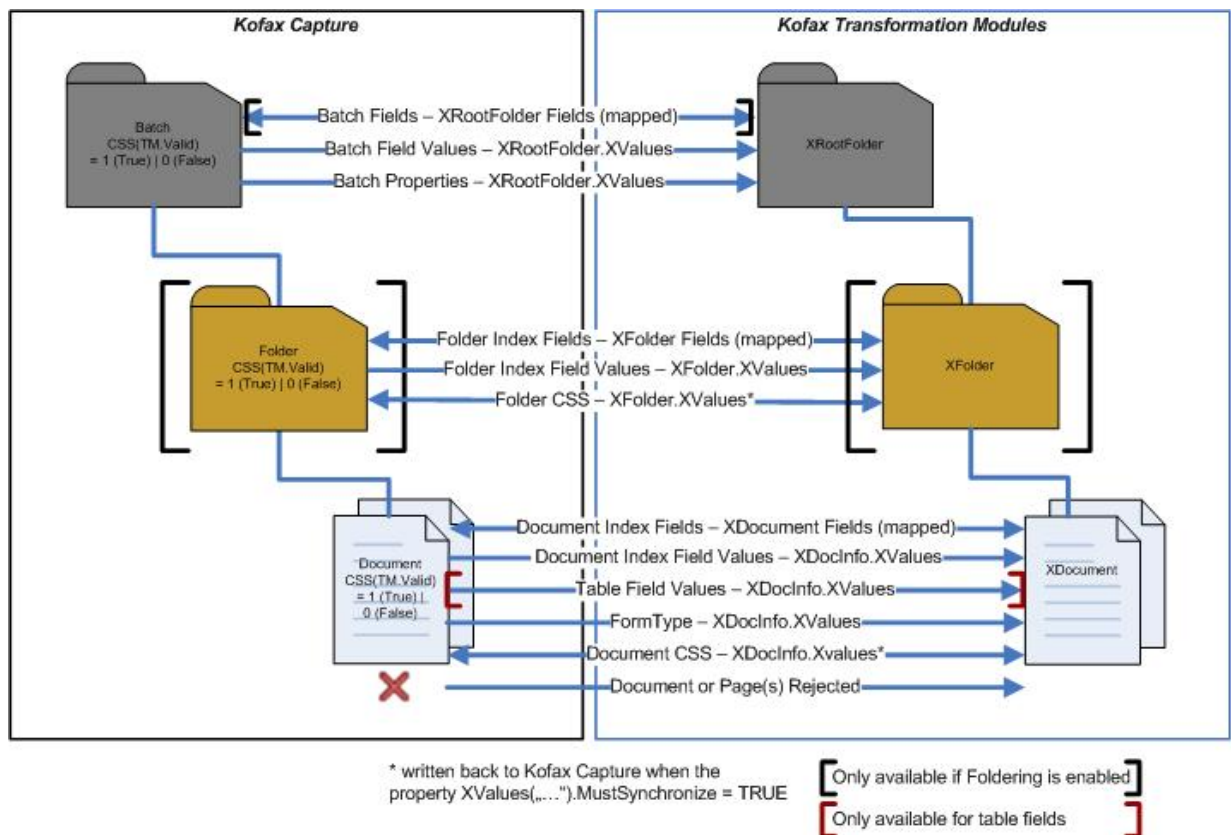


Event of Multi-Field Script Validation Method

## Access to Kofax Capture Data

XValues are used to make more information about the corresponding Kofax Capture objects available in script. The following figure shows the data transfer between Kofax Capture and Kofax Transformation Modules. The following applies to the Kofax Capture integration of this product only.





## Data transfer between Kofax Capture objects and the integration

### Batch Data

- Batch Fields (mapped)

If folders are enabled the batch fields can be mapped with folder fields of the root XFolder. Then the access is read/write during runtime and when updating the folder field of the root XFolder, the data are written to the mapped batch field on Kofax Capture side.

- Batch Field Values

All batch field values are accessible read-only from script. They are saved in the corresponding XValue of the root XFolder.

```
XRootFolder.XValues("AC_FIELD_" & BatchFieldName)
```

- Batch Properties

Additionally the following batch properties are accessible with read-only access from script:

Name	XRootFolder.XValues("AC_BATCH_NAME")
Batch Class Name	XRootFolder.XValues("AC_BATCH_CLASS_NAME")
Priority	XRootFolder.XValues("AC_BATCH_PRIORITY")
ImageDirectory	XRootFolder.XValues("AC_BATCH_DIRECTORY")
ExternalBatchID	XRootFolder.XValues("AC_BATCH_EXTERNAL_BATCH")
BatchGUID	XRootFolder.XValues("AC_BATCH_GUID")

BatchCreationDateTime	XRootFolder.XValues("AC_BATCH_CREATIONDATETIME")
CreationSiteName	XRootFolder.XValues("AC_BATCH_CREATIONSITENAME")
CreationUserID	XRootFolder.XValues("AC_BATCH_CREATIONUSERID")
OperatorUserID (user ID of last batch history entry)	XRootFolder.XValues("AC_BATCH_OPERATORUSERID")

TODO: Valid / Verified

### Multiple Steps of Validation

If for the validation the multiple steps are activated the information about the next validation step that is required for validating the batch can be found in the CustomStorageString `TM.BatchNextValidationStep` as a one-based number. This information can be used for example for workflow agents that route the batches and skip certain Validation steps

TODO not working -> not correctly implemented.

## Folder Data

Folder fields and folders are only available when folders have been enabled in the project settings. Having folders enabled, one Kofax Capture folder class is mapped to one folder definition on the Kofax Transformation Modules integration side. The folders are assigned to a fixed level in Kofax Transformation, this concept is inherited by the folder class at the Kofax Capture although the folder class does not have this limitation.

- Folder Index Fields (mapped)

All mapped folder fields have read/write access and by updating the folder field of the XFolder its data are written to the mapped Kofax Capture folder field. The initial values of the folder fields are read from the mapped Kofax Capture folder field.

- Folder Index Field Values

During runtime all folder field values of a folder are accessible read-only from script. They are saved in the corresponding XValue of the XFolder object.

```
XFolder.XValues("AC_FIELD_" & FolderFieldName)
```

- Folder CSS

Additionally, all CustomStorageStrings defined on the Kofax Capture side are transferred to the XValues collection of the XFolder object.

```
XFolder.XValues("AC_CSS_" & CSSName)
```

In order to write CustomStorageString values back to Kofax Capture, set the property "MustSynchronize" to TRUE so that changes in the XValues are transferred to the corresponding CustomStorageString. By using the naming convention and setting the "MustSynchronize" property you can also add new XValues to the collection of the XFolder so that new CustomStorageStrings on Kofax Capture side are created.

TODO: Valid / Verified

## Document Data

- Document Index Fields (mapped)

For all mapped document index fields you have read/write access. By updating the document field of the XDocInfo.XDocument object its data is written to the mapped Kofax Capture document index field. Initial field values from Kofax Capture document index fields are not considered.

Mapped table fields have read/write access and by updating the table of the XDocInfo.XDocument object its data are written to the mapped Kofax Capture table field.

- Document Index Field Values

During runtime all document index field values are accessible read-only from script. They are saved in the corresponding XValue of the XDocInfo object. Default values defined in Kofax Capture for document index fields are not filled until the validation step in the processing queue. The corresponding XValues remain empty during server processing.

```
XDocInfo.XValues("AC_FIELD_" & FieldName)
```

- Table Field Values

During runtime all table fields values are accessible read-only from script. They are saved row by row in the XValues of the XDocInfo object. The row number is zero-based and the columns are separated by [Tab]. See below a script example.

```
XDocInfo.XValues("AC_TABLE_" & TableFieldName & "_ROW_" & nRowNumber)
```

- Form Type

The Kofax Capture form type name can be accessed read-only using the following key:

```
XDocInfo.XValues("AC_FORMTYPE")
```

- Document CSS

Additionally, all CustomStorageStrings defined on the Kofax Capture side are transferred to the XValues collection of the XDocInfo object. .

```
XDocInfo.XValues("AC_CSS_" & CSSName)
```

In order to write values back to Kofax Capture, set the property MustSynchronize to TRUE so that changes in the XValue are transferred to the corresponding document's CustomStorageString. By using the naming convention and setting the property MustSynchronize to TRUE you can also add new XValues to the collection of the XDocInfo so that new CustomStorageStrings on Kofax Capture side are created.

- Document or Page(s) Rejected

If a document or one of its pages is rejected in Kofax Capture is reflected in the XValues of the document.

The following XValue exists if a document is rejected:

```
XDocInfo.XValues("AC_REJECTED_DOCUMENT")
```

By using the following key you get the rejection note for a document if it is existing:

```
XDocInfo.XValues("AC_REJECTED_DOCUMENT_NOTE")
```

If one of the documents' pages is rejected you find per page an XValue in the document with the following key:

```
XDocInfo.XValues("AC_REJECTED_PAGE<n>")
```

Where <n> is the one-based page number of the page you are interested in. To get the rejection note of a page get the value of the XValue by the following key:

```
XDocInfo.XValues("AC_REJECTED_PAGE_NOTE<n>")
```

TODO: Valid / Verified

**Example. Accessing the table field rows of a Kofax Capture table field in the XValues of an XDocInfo object:**

```
Private Sub Document_BeforeExtract(pXDoc As CASCADELib.CscXDocument)
```

```

Dim sRow As String
Dim n As Long
Dim sCols() As String

' first row index is zero
n = 0

While pXDoc.XValues.ItemExists("AC_TABLE_MyTableName_Row_" & n) = True

    sRow = pXDoc.XValues("AC_TABLE_MyTableName_Row_" & n).Value
    sCols = Split(sRow, vbTab)

    ' ...
    ' use the column results for your needs
    ' ...

    n = n + 1

Wend

End Sub

```

### Example. Checking if a document or one of its pages is rejected in Kofax Capture:

```

Private Function IsDocumentRejected(oXDocInfo As CASCADELib.CscXDocInfo) As Boolean
    IsDocumentRejected = False
    If oXDocInfo.XValues.ItemExists("AC_REJECTED_DOCUMENT") Then
        IsDocumentRejected = True
    Else
        Dim nPageIndex As Long
        For nPageIndex = 0 To oXDocInfo.PageCount - 1
            If oXDocInfo.XValues.ItemExists("AC_REJECTED_PAGE" & CStr(nPageIndex + 1)) Then
                IsDocumentRejected = True
            End If
        Next
    End If
End Function

```

### Example. Getting the rejection note in Kofax Capture of a document or one of its pages:

```

Private Function GetRejectionNote(oXDocInfo As CASCADELib.CscXDocInfo, Optional nPageIndex = 0
    As Long) As String

    GetRejectionNote = ""

    If nPageIndex = 0 Then
        ' get the rejection note of the document
        If oXDocInfo.XValues.ItemExists("AC_REJECTED_DOCUMENT_NOTE") Then
            GetRejectionNote = oXDocInfo.XValues("AC_REJECTED_DOCUMENT_NOTE")
        End If
    Else
        ' get the rejection note of the page, nPageIndex is one-based
        If oXDocInfo.XValues.ItemExists("AC_REJECTED_PAGE_NOTE" & CStr(nPageIndex)) Then
            GetRejectionNote = oXDocInfo.XValues("AC_REJECTED_PAGE_NOTE" & CStr(nPageIndex))
        End If
    End If
End Function

```

## Folding

The general description of folding is done in the chapter about the [RootFolder](#) object that is handling the events for the folding implementation.

Folding allows you to bracket documents which are logically belonging together, like documents sent by one customer. The folder fields can be used to summarize document data like invoice totals to have an overview about the total of a batch of if the documents are grouped by supplier the total of the invoices per supplier.

### Example. 1. Grouping Documents by Field Value

Creates a folder for all documents with the same value in the field "DepotNumber" and adds the documents to it.

```
' extraction script for folder Root
' ...
Private Sub RootFolder_DoFoldering(pXRootFolder As CASCADELib.CscXFolder)

    If Project.ScriptExecutionMode = CscScriptModeValidation then exit sub

    Dim i As Long
    For i = pXRootFolder.DocInfos.Count - 1 To 0 Step -1
        Dim pFolder As CscXFolder
        Dim DepotNumber As String
        If pXRootFolder.DocInfos(i).XDocument.Fields.Exists("DepotNumber") Then
            DepotNumber =
pXRootFolder.DocInfos(i).XDocument.Fields.ItemByName("DepotNumber").Text
        Else
            DepotNumber = ""
        End If
        Set pFolder = FindFolder(pXRootFolder, DepotNumber)
        If pFolder Is Nothing Then
            Set pFolder = New CscXFolder
            pXRootFolder.Folders.Add(pFolder)
            pFolder.Fields.ItemByName("DepotNumber").Text = DepotNumber
        End If
        pFolder.DocInfos.MoveInto pXRootFolder.DocInfos(i), 0
    Next i
End Sub
Private Function FindFolder(pRootFolder As CscXFolder, Depotnummer As String) As CscXFolder
    Dim i As Long

    For i = 0 To pRootFolder.Folders.Count - 1
        If pRootFolder.Folders(i).Fields.ItemByName("Depotnummer").Text = Depotnummer Then
            Set FindFolder = pRootFolder.Folders(i)
            Exit Function
        End If
    Next i

    Set FindFolder = Nothing
End Function
```

### Example. 2. Creating Folder per Document

Creates a sub folder for each document and moves the document to it.

```
' extraction script for folder Root
' ...
Private Sub RootFolder_DoFoldering(pXRootFolder As CASCADELib.CscXFolder)

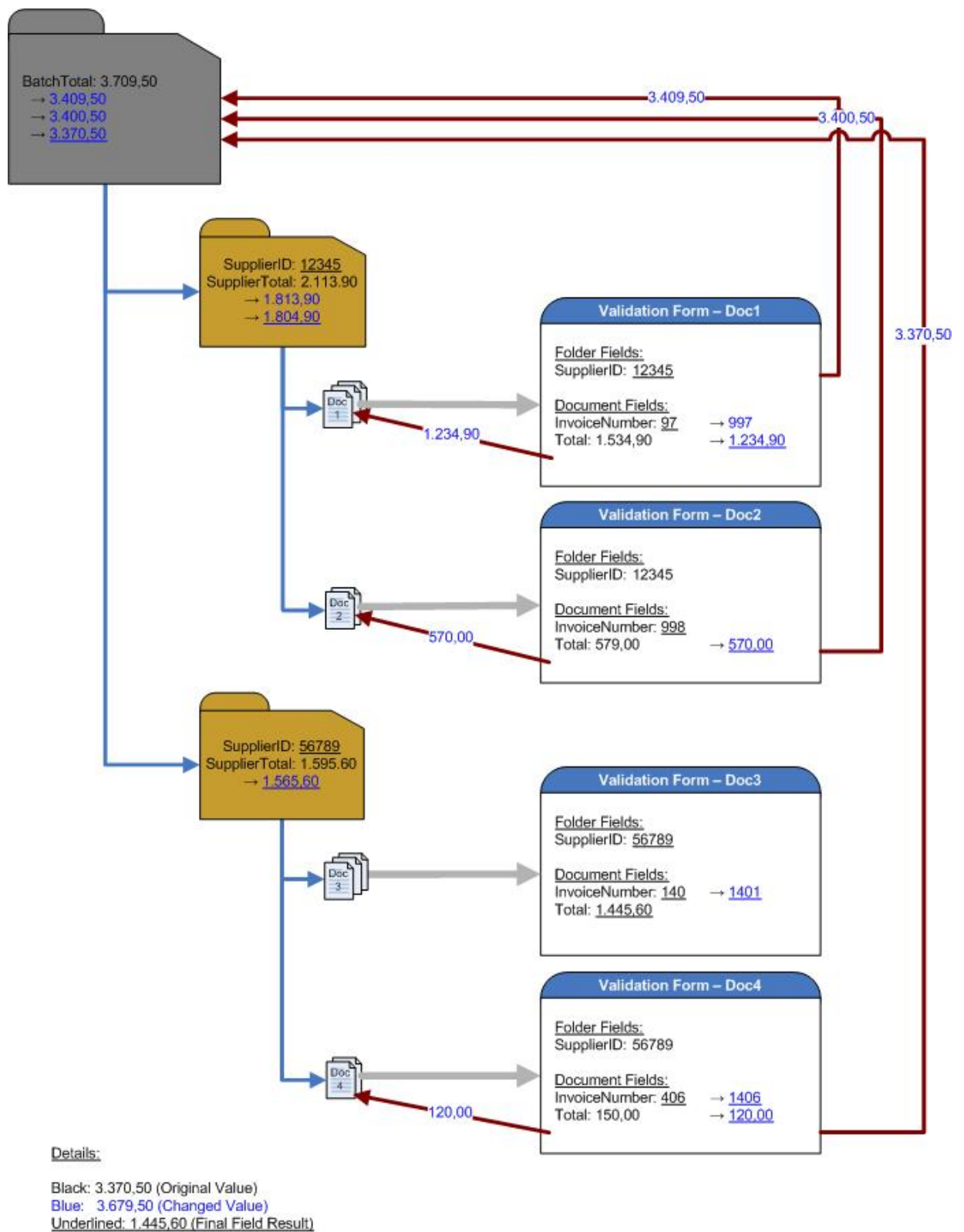
    If Project.ScriptExecutionMode = CscScriptModeValidation then exit sub

    Dim i As Long
    Dim oXDocInfo As CscXDocInfo
    Dim oSubXFolder As CscXFolder
    ' we remove documents from the head of the list
    ' therefore we need a while loop here
```

```
While pXRootFolder.DocInfos.Count > 0
  Set oXDocInfo = pXRootFolder.DocInfos(0)
  ' create new sub folder for each document
  Set oSubXFolder = New CscXFolder
  pXRootFolder.Folders.Add oSubXFolder
  'move current document into new subfolder
  oSubXFolder.DocInfos.MoveInto oXDocInfo, oSubXFolder.DocInfos.Count
Wend
End Sub
```

## Summarizing Field Data

The foldering can be used to summarize data at different levels. In this example the invoice total is summarized per supplier and for the complete batch. You can find the script needed for the summarization below the figure showing a batch with two folders having each two documents.



### Example. Folder Extraction Script

```
Private Sub Folder_AfterExtract(pXFolder As CASCADELib.CscXFolder)
```

```

Dim n As Integer
Dim dSupplierTotal As Double

For n = 0 To pXFolder.DocInfos.Count-1
    dSupplierTotal = dSupplierTotal +
CDBl(pXFolder.DocInfos(n).XDocument.Fields("Total").Text)
Next n

pXFolder.Fields.ItemByName("SupplierTotal").Text = Format(dSupplierTotal, "#.###,00")

End Sub

```

### Example. ValidationForm Script

```

Private Sub ValidationForm_AfterFieldConfirmed(pXDoc As CASCADELib.CscXDocument, pField As
CASCADELib.CscXDocField)

    If pField.Name = "Total" Then
        XDoc.ParentFolder.Fields.ItemByName("SupplierTotal").Text =
Format(RecalculateTotal(pXDoc), "#.###,00")
    End If

End Sub

Private Function RecalculateTotal(pXDoc As CASCADELib.CscXDocument) As Double
    Dim oXDocInfo As CASCADELib.CscXDocInfo
    Dim n As Integer
    RecalculateTotal = 0
    For n = 0 To pXDoc.ParentFolder.DocInfos.Count - 1
        Set oXDocInfo = pXDoc.ParentFolder.DocInfos(n)
        RecalculateTotal = RecalculateTotal + oXDocInfo.XDocument.Fields.ItemByName("Total")
    Next n

End Function

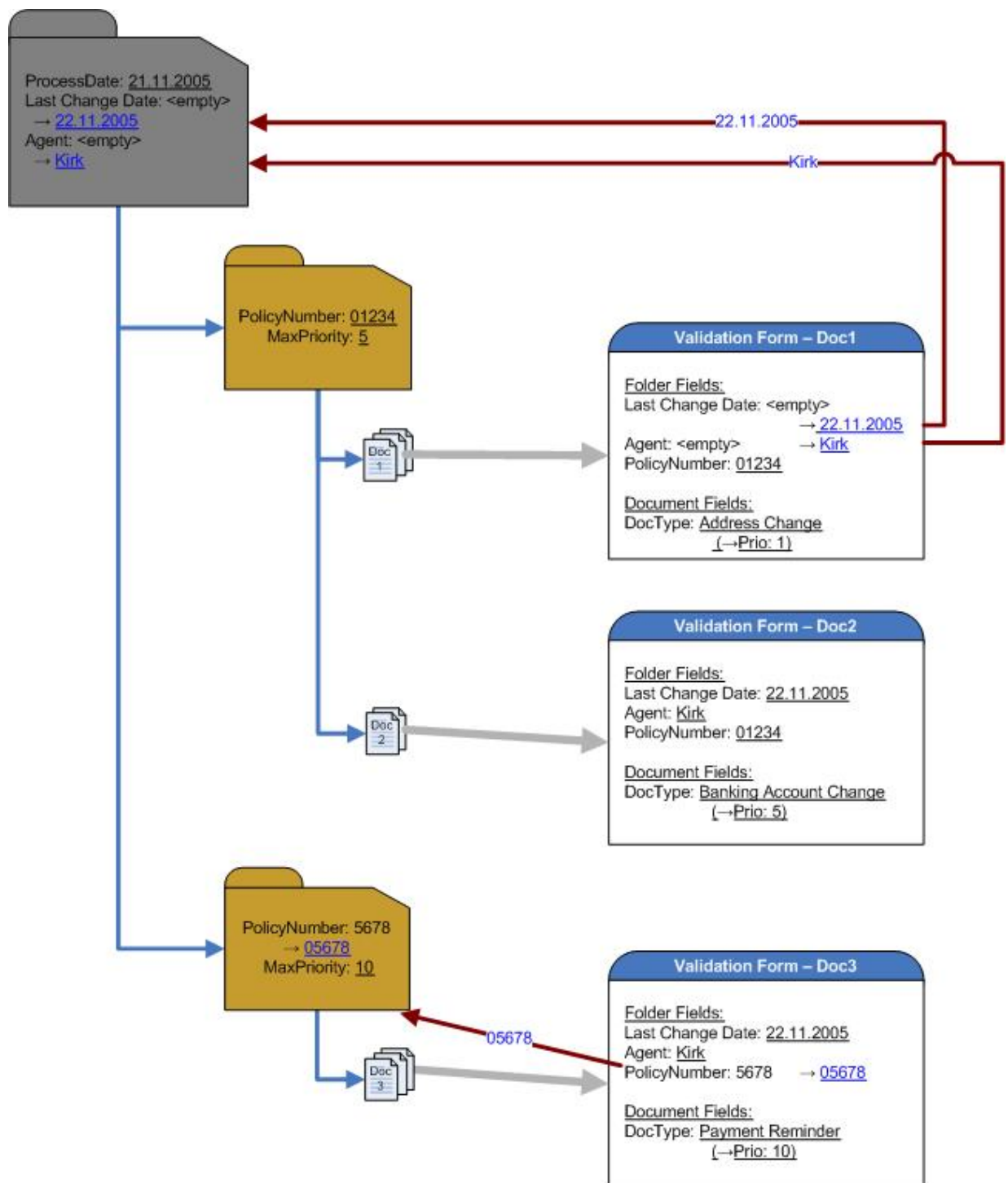
```

## Changing Folder and Document Field Data

The following figure shows the inheritance of folder field data in a simple scenario. The incoming documents are classified and the document type is determined. Depending on the document type a priority is set in script. If a document is reclassified, the priority is adjusted. The maximum of the document's priorities is set as the folder field's maximum priority. The Agent and the Last Change Date are defined on batch level and are thus modified only once, applying to all documents contained in the batch.

The Policy Number is an invisible document field; the initial value is what it was given when the folder was initially created and its value is assigned to a corresponding folder field. Updating this folder field thus affects all documents that are part of the folder.



Details:

Black: 11.11.2005 (Original Value)

Blue: 18.11.2005 (Changed Value)

Underlined: 21.11.2005 (Final Field Result)**Sample for the inheritance of folder field data**

**Example. Project Script**

```

Private Sub ValPolicyNumber_ValidateFolderFields(ValItems As CASCADELib.CscXDocValidationItems,
_
pXFolder As CASCADELib.CscXFolder, ErrDescription As String, ValidField As Boolean)
    Dim strPolicyNumber As String
    Dim i As Long

    ValidField = True

    For i = 0 To pXFolder.DocInfos.Count - 1
        If i = 0 Then
            strPolicyNumber =
pXFolder.DocInfos(i).XDocument.Fields.ItemByName("PolicyNumber").Text
            If pXFolder.Fields.ItemByName("PolicyNumber").Text <> strPolicyNumber Then
                pXFolder.Fields.ItemByName("PolicyNumber").Text = strPolicyNumber
            End If
        Else
            If pXFolder.DocInfos(i).XDocument.Fields.ItemByName("PolicyNumber").Text <>
strPolicyNumber Then
                ValidField = False
                ErrDescription = "Folder contains different policy numbers."
                Exit For
            End If
        End If
    Next i
End Sub

```

**Example. Root Folder Script**

```

Private Sub Batch_DoFoldering(pXRootFolder As CASCADELib.CscXFolder)

    Dim i As Long
    Dim oXDoc As CscXDocument

    For i = pXRootFolder.DocInfos.Count - 1 To 0 Step -1
        Dim pFolder As CscXFolder
        Dim sPolicyNumber As String

        Set oXDoc = pXRootFolder.DocInfos(i).XDocument

        ' find existing folder having the PolicyNumber or create a new one
        sPolicyNumber = oXDoc.Fields.ItemByName("PolicyNumber").Text
        Set pFolder = FindFolder(pXRootFolder, sPolicyNumber)
        If pFolder Is Nothing Then
            Set pFolder = New CscXFolder
            pXRootFolder.Folders.Insert(pFolder, 0)
            pFolder.Fields.ItemByName("PolicyNumber").Text = sPolicyNumber
            pFolder.Fields.ItemByName("PolicyNumber").ExtractionConfident=True
        End If

        ' move document into found or new folder
        pFolder.DocInfos.MoveInto (pXRootFolder.DocInfos(i), 0)

    Next i
End Sub

Private Function FindFolder(pRootFolder As CscXFolder, sPolicyNumber As String) As CscXFolder
    Dim i As Long

    For i = 0 To pRootFolder.Folders.Count - 1
        If pRootFolder.Folders(i).Fields.ItemByName("PolicyNumber").Text = sPolicyNumber Then
            Set FindFolder = pRootFolder.Folders(i)
            Exit Function
        End If
    End If
End Function

```

```

Next i

    Set FindFolder = Nothing

End Function

Private Sub Folder_BeforeExtract(pXFolder As CASCADELib.CscXFolder)
    If Project.ScriptExecutionMode = CscScriptModeServerDesign Or
       Project.ScriptExecutionMode=CscScriptModeServer Then
        pXFolder.Fields.ItemByName("ProcessDate").Text = Format(Now(),"Short Date")
    End If
End Sub

```

### Example. Folder Script (PolicyNumber)

```

Private Sub Folder_BeforeExtract(pXFolder As CASCADELib.CscXFolder)

    Dim nMaxPriority As Long
    Dim nPriority As Long
    Dim i As Long

    For i = 0 To pXFolder.DocInfos.Count - 1
        nPriority = CInt(pXFolder.DocInfos(i).XDocument.Fields.ItemByName("Priority").Text)
        If nPriority > nMaxPriority Then
            nMaxPriority = nPriority
        End If
    Next i

    pXFolder.Fields.ItemByName("MaxPriority").Text = CStr(nMaxPriority)
    pXFolder.Fields.ItemByName("MaxPriority").ExtractionConfident = True

End Sub

```

### Example. Class Script (Base)

```

Private Sub Document_AfterExtract(pXDoc As CASCADELib.CscXDocument)
    Select Case pXDoc.Fields.ItemByName("DocType").Text
        Case "Address Change"
            pXDoc.Fields.ItemByName("Priority").Text = "1"
        Case "Payment Reminder"
            pXDoc.Fields.ItemByName("Priority").Text = "10"
        Case "Banking Account Change"
            pXDoc.Fields.ItemByName("Priority").Text = "5"
    End Select
    pXDoc.Fields.ItemByName("Priority").ExtractionConfident = True
End Sub

Private Sub ValidationForm_AfterComboboxItemSelected(ByVal ComboboxName As String, _
pXDoc As CASCADELib.CscXDocument, ByVal SelectedIndex As Long, ByVal SelectedText As String)
    Select Case ComboboxName
        Case "DocType"
            Select Case SelectedText
                Case "Address Change"
                    pXDoc.Fields.ItemByName("Priority").Text = "1"
                Case "Payment Reminder"
                    pXDoc.Fields.ItemByName("Priority").Text = "10"
                Case "Banking Account Change"
                    pXDoc.Fields.ItemByName("Priority").Text = "5"
            End Select
        End Select
    End Sub

Private Sub ValidationForm_BeforeComboBoxDropDown(ByVal ComboboxName As String, pXDoc As
CASCADELib.CscXDocument, ComboboxItems As String)

```

```

Select Case ComboboxName
    Case "DocType"
        ComboboxItems = "Payment Reminder;Banking Account Change;Address Change"
End Select
End Sub

```

## Batch Restructuring

Batch restructuring enables changes to be applied to the batch structure using the methods and functions of the [Batch](#) object. It is allowed in `Batch_Open`, `Batch_Close` event and additionally if folders are enabled in the `RootFolder_DoFoldering` event. The batch events are implemented in the project script sheet and can be tested during design time using the dialog Test Runtime Script Events. The `RootFolder_DoFoldering` is placed in the root folder script sheet and can be tested by calling Perform Auto-Foldering in the test documents.

Batch restructuring may have impacts on the states of the involved objects. Most of these operations switch the state of these objects to invalid. Using e.g. the method `Batch.DeleteDocument` makes the root `XFolder` invalid. This can be ignored if the batch is no more showing up in any module. By default the validation of all changed objects is executed again to care for the consistency of the data. If these are expensive operations and if they are not needed this can be suppressed by setting before the execution of the batch restructuring methods the property `Batch.ValidateAfterBatchRestructuring` to `FALSE`.

If batch restructuring is applied in the `Batch_Open` event of server processing the defined validation rules will be applied anyway. If it is applied at the end of server processing the validation has to be reexecuted again. The same is true if batch restructuring is applied in one of the server following workflow steps.

---

**Important** If applying batch restructuring at a later point than server processing you have to care for not doing operations that lead to object states that are not valid for the current workflow step.

At any case you have to be aware that when applying batch restructuring in the user interactive modules the opening or the closing of a batch may be slowed down.

---



---

**Note** Applying the restructuring functionality does not fire the events related to the similar actions of batch editing.

---

## Page Operations

The following restructuring operations on the [Batch](#) object are available for the pages. Parameters are in parentheses.

### Delete Page

`Batch.DeletePage (ByVal pXDocInfo As CscXDocInfo, ByVal PageIndex As Long)`

Calling this method the specified page is deleted from the given document (*pXDocInfo*). The document the page is deleted from is initially getting invalid.

It is not possible to delete the last page from a document, use for this the [DeleteDocument](#) method.

All field results that are attached to the deleted page are removed from the field. The classification result of the document is kept.

### Example. Deleting pages with an even index of a document

```

Private Sub DeletePagesWithEvenIndex(oXDocInfo As CASCADELib.CscXDocInfo)

    Dim i As Long
    Dim PageIndex As Long
    If oXDocInfo.PageCount > 1 Then
        For PageIndex = oXDocInfo.PageCount - 1 To 0 Step -1
            If (PageIndex + 1) Mod 2 = 0 Then
                Batch.DeletePage(oXDocInfo, PageIndex)
            End If
        Next PageIndex
    End If

End Sub

```

## Move Page

```
Batch.MovePage(ByVal pXDocInfo As CscXDocInfo, ByVal FromPageIndex As Long,
ByVal ToPageIndex As Long)
```

```
Batch.MovePageTo(ByVal pFromXDocInfo As CscXDocInfo, ByVal FromPageIndex As
Long, ByVal pToXDocInfo As CscXDocInfo, ByVal ToPageIndex As Long)
```

A page can be moved inside a document (*pXDocInfo*) or via *MovePageTo* moved to another document (*pFromXDocInfo, pToXDocInfo*). All data attached is moved along with that page such as OCR results or field results that are placed on that page. When moving the page to a different document the data attached to this page is no longer available in the source document, so fields may lose their results. Initially the document where a page has been moved internally or both documents the source and the destination document if moving outside are getting invalid. The classification result of the document persists.

### Example. Moving the cover sheet to the end of the document

```

Private Sub MoveCoverSheet(oXDocInfo As CASCADELib.CscXDocInfo)

    If oXDocInfo.PageCount > 1 Then
        Batch.MovePage(oXDocInfo, 0, oXDocInfo.PageCount)
    End If

End Sub

```

## Copy Page

```
Batch.CopyPage(ByVal pXDocInfo As CscXDocInfo, ByVal FromPageIndex As Long,
ByVal ToPageIndex As Long)
```

```
Batch.CopyPageTo(ByVal pFromXDocInfo As CscXDocInfo, ByVal FromPageIndex As
Long, ByVal pToXDocInfo As CscXDocInfo, ByVal ToPageIndex As Long)
```

A page (*FromPageIndex*) can be copied inside of a document (*pXDocInfo*) or via *CopyPageTo* copied to another document (*pToXDocInfo*) by specifying its new position (*ToPageIndex*). The data attached to the copied page is duplicated and attached to the copied pages. The initial status of the target document is invalid. The classification result of the target document persists.

### Example. Copying first page of first document to all other documents of a batch

```

Private Sub CopyFirstPage(pXRootFolder As CASCADELib.CscXFolder)

    Dim oFirstXDocInfo As CASCADELib.CscXDocInfo
    Set oFirstXDocInfo = pXRootFolder.GetDocInfoByGlobalIndex(0)

```

```

Dim i As Long
For i = 1 To pXRootFolder.GetTotalDocumentCount-1
    Dim oXDocInfo As CASCADELib.CscXDocInfo
    Set oXDocInfo = pXRootFolder.GetDocInfoByGlobalIndex(i)
    Batch.CopyPageTo(oFirstXDocInfo, 0, oXDocInfo, 0)
Next
End Sub

```

## Document Operations

The following restructuring operations on the [Batch](#) object are available for the document object.

### Delete Document

```
Batch.DeleteDocument(ByVal pXFolder As CscXFolder, ByVal DocIndex As Long)
```

Calling this method the specified document is deleted from the given folder (*pXFolder*) containing that document.

Initially the parent folder of the deleted document is getting invalid.

#### Example. Delete the first document from a batch

```

Private Sub DeleteFirstDocFromFolder(pXFolder As CASCADELib.CscXFolder)
    Batch.DeleteDocument(pXRootFolder, 0)
End Sub

```

### Copy Document

```
Batch.CopyDocument(ByVal pXFolder As CscXFolder, ByVal FromDocIndex As Long,
ByVal ToDocIndex As Long)
```

```
Batch.CopyDocumentTo(ByVal pFromXFolder As CscXFolder, ByVal FromDocIndex As
Long, ByVal pToXFolder As CscXFolder, ByVal ToDocIndex As Long)
```

A document can be copied inside of a folder (*pXFolder*) or via `CopyDocumentTo` to another folder (*pFromXFolder*, *pToXFolder*) by specifying its new position in the target folder (*ToDocIndex*). The copied document is a duplicate of the original one and is initially invalid.

TODO: Classification Result

#### Example. Copy the first document to all subfolders

```

Private Sub CopyFirstDocToSubfolders(pXRootFolder As CASCADELib.CscXFolder)

    Dim i As Long
    For i = 1 To pXRootFolder.Folders.Count - 1
        Dim oXFolder As CASCADELib.CscXFolder
        Set oXFolder = pXRootFolder.Folders(i)
        Batch.CopyDocumentTo(pXRootFolder, 0, oXFolder, 0)
    Next
End Sub

```

### Move Document

```
Batch.MoveDocument(ByVal pXFolder As CscXFolder, ByVal FromIndex As Long,
ByVal ToIndex As Long)
```

```
Batch.MoveDocumentTo(ByVal pFromXFolder As CscXFolder, ByVal FromIndex As
Long, ByVal pToXFolder As CscXFolder, ByVal ToIndex As Long)
```

A document can be moved inside a folder (*pXFolder*) or via `MoveDocumentTo` also to another folder (*pFromXFolder*, *pToXFolder*) by specifying its new position in the target folder (*ToIndex*). Initially all involved folders are getting invalid.

### Example. Moving the first document to the end of a batch

```
Private Sub MoveFirstDocToEndBatch(pXRootFolder As CASCADELib.CscXFolder)

    Dim i As Long
    For i = 1 To pXRootFolder.Folders.Count - 1
        Dim oXFolder As CASCADELib.CscXFolder
        Set oXFolder = pXRootFolder.Folders(i)
        Batch.CopyDocumentTo(pXRootFolder, 0, oXFolder, 0)
    Next

End Sub
```

### Split Document

```
Batch.SplitDocument(ByVal pXDocInfo As CscXDocInfo, ByVal PageIndex As Long)
```

A document (*pXDocInfo*) can be split into two documents. All data attached to the pages stays with the page it belongs to. The new document is inserted behind the source document and it inherits the classification result from the source document. This can be changed by applying the [ChangeClass](#) method.

### Example. Split after first page of first document

```
Private Sub SplitAfterFirstPage(oXDocInfo As CASCADELib.CscXDocInfo)

    Batch.SplitDocument(oXDocInfo, 0)

End Sub
```

### Create Document

```
Batch.CreateDocumentFromPage (pXDocInfo As CscXDocInfo, PageIndex As Long) As Long
```

```
Batch.CreateDocumentFromPageTo (pXDocInfo As CscXDocInfo, PageIndex As Long,
pToXFolder As CscXFolder, ToDocIndex As Long) As Long
```

A document can be created from a specified page (*PageIndex*) and it will be added to the end of the document collection the source document (*pXDocInfo*) belongs to. By calling `CreateDocumentFromPageTo` also the target folder (*pToXFolder*) and the position of the created document can be specified. The page the document is created from will be removed from the source document. The returned index is the position of the newly created document. These methods are applicable when having more than one page.

The new document inherits the classification result from the source document, you can change this by applying the [ChangeClass](#) method. Initially both documents, the source and the created one, are getting invalid.

TODO: Classification Result

### Example. TODO

TODO

### Change Class

```
Batch.ChangeClass(ByVal ClassName As String, ByVal pXDocInfo As CscXDocInfo,
ByVal bValidate As Boolean)
```

The class a document is assigned to can be changed by applying this method. The passed name of the class (*ClassName*) must be a valid classification result. Existing document data might be lost during this operation, if the fields are not existing in the new class, else the results are retained. The document gets initially invalid. The validation of the document can be directly forced by setting the flag *bValidate* to TRUE, then all the validation rules are applied immediately.

#### Example. TODO

```
Batch.ChangeClass("MyNewClass", pXRootFolder.DocInfos(0), False)
```

### Modifying Document Fields

```
oXDocInfo.XDocument.Fields.ItemByName("MyField").Text = "MyNewText"
```

During batch restructuring it is allowed to change document fields. These changes are synchronized back to Kofax Capture. By default the validation of the batch is re-executed after batch restructuring so that the fields are checked against the defined formatting and validation rules.

#### Example. TODO

TODO

## Folder Operations

The following restructuring operations on the [Batch](#) object are available for the folder object.

### Delete Folder

```
DeleteFolder(ByVal pParentXFolder As CscXFolder, ByVal FolderIndex As Long)
```

Deletes the specified folder (*FolderIndex*) from its parent folder (*pParentXFolder*) with all sub folders and documents inside. Initially the parent folder is getting invalid.

#### Example. Delete first folder

```
Private Sub DeleteFirstFolder(pXRootFolder As CASCADELib.CscXFolder)
    Batch.DeleteFolder(pXRootFolder, 0)
```

```
End Sub
```

### Move Folder

```
MoveFolder(ByVal pParentXFolder As CscXFolder, ByVal FromIndex As Long, ByVal
ToIndex As Long)
```

```
MoveFolderTo(ByVal pFromParentXFolder As CscXFolder, ByVal FromIndex As Long,
ByVal pToParentXFolder As CscXFolder, ByVal ToIndex As Long)
```



A folder can be moved inside its parent folder (*pParentXFolder*), or can be moved to another parent folder as long as it stays on the same folder level. Initially the move operation is making all involved parent folders invalid.

#### Example. Move first folder to end of folder list of parent folder

```
Private Sub MoveFirstFolderToEnd(pParentXFolder As CASCADELib.CscXFolder)

    Batch.MoveFolder(pParentXFolder, 0, pParentXFolder.DocInfos.Count)

End Sub
```

#### Split Folder

```
Batch.SplitFolderBeforeDocument(ByVal pParentXFolder As CscXFolder, ByVal
DocIndex As Long)

Batch.SplitFolderBeforeFolder(ByVal pParentXFolder As CscXFolder, ByVal
FolderIndex As Long)
```

A folder (*pParentXFolder*) can be split before a specified document or before a specified folder. If splitting before a document one folder contains all documents before the specified document and the other one the specified document, all documents after that and all sub folders if present. If splitting before a folder, one folder contains all documents of the original folder and all sub folders before the specified one and the other folder the specified folder and all following sub folders if present. Initially the split folder is getting invalid.

#### Example. Split first folder before first sub folder

```
Private Sub SplitFirstFolder(pParentXFolder As CASCADELib.CscXFolder)

    Batch.SplitFolderBeforeFolder(pParentXFolder, 0)

End Sub
```

#### Create Folder

```
Batch.CreateFolder(ByVal pParentXFolder As CscXFolder) As Long

Batch.CreateFolderFromDocument(ByVal pParentXFolder As CscXFolder, ByVal
DocIndex As Long) As Long
```

A folder can be created from an existing document or can be created as an empty folder in the specified parent folder (*pParentXFolder*) and documents can be moved into it afterwards. The level the folder is created for must exist. Initially the parent folder and the created folder are getting invalid.

#### Example. Create a folder and return it

```
Private Function CreateFolder(pParentXFolder As CASCADELib.CscXFolder) As CASCADELib.CscXFolder

    Dim nFolderIndex As Long
    nFolderIndex = Batch.CreateFolder(pParentXFolder)
    Set CreateFolder = pParentXFolder.Folders.ItemByIndex(nFolderIndex)

End Sub
```

#### Modifying Folder Fields

```
oXFolder.Fields.ItemByName("MyField").Text = "MyNewText"
```

During batch restructuring it is allowed to change folder fields. These changes are synchronized back to Kofax Capture. By default the validation of the batch is re-executed after batch restructuring so that the fields are checked against the defined formatting and validation rules.

### Example. TODO

TODO

## Document Routing

Document Routing enables you to move documents from a parent to a child batch and assign these batches to any module in the defined Kofax Capture Workflow to process the parent or child batch via script. The child batch, inherits the settings from the parent batch.

---

**Note** Document Routing is available only with Kofax Capture Version 9 or later.

---

The functionality is defined by setting special [XValues](#) that are analyzed and applied after the `Batch_Close` event. Document Routing is available for all modules that support scripting. The XValues at the [XDocInfo](#) (document) level determine if a document is moved (implicitly creating a child batch). The XValues at the [root XFolder](#) level define the next module to process both parent and child batches.

---

**Note** Document Routing runs after the `Batch_Close` event is fired as the result of analyzing and applying the information collected in the XValues during batch processing. Afterwards, all XValues that are related to Document Routing are deleted.

---



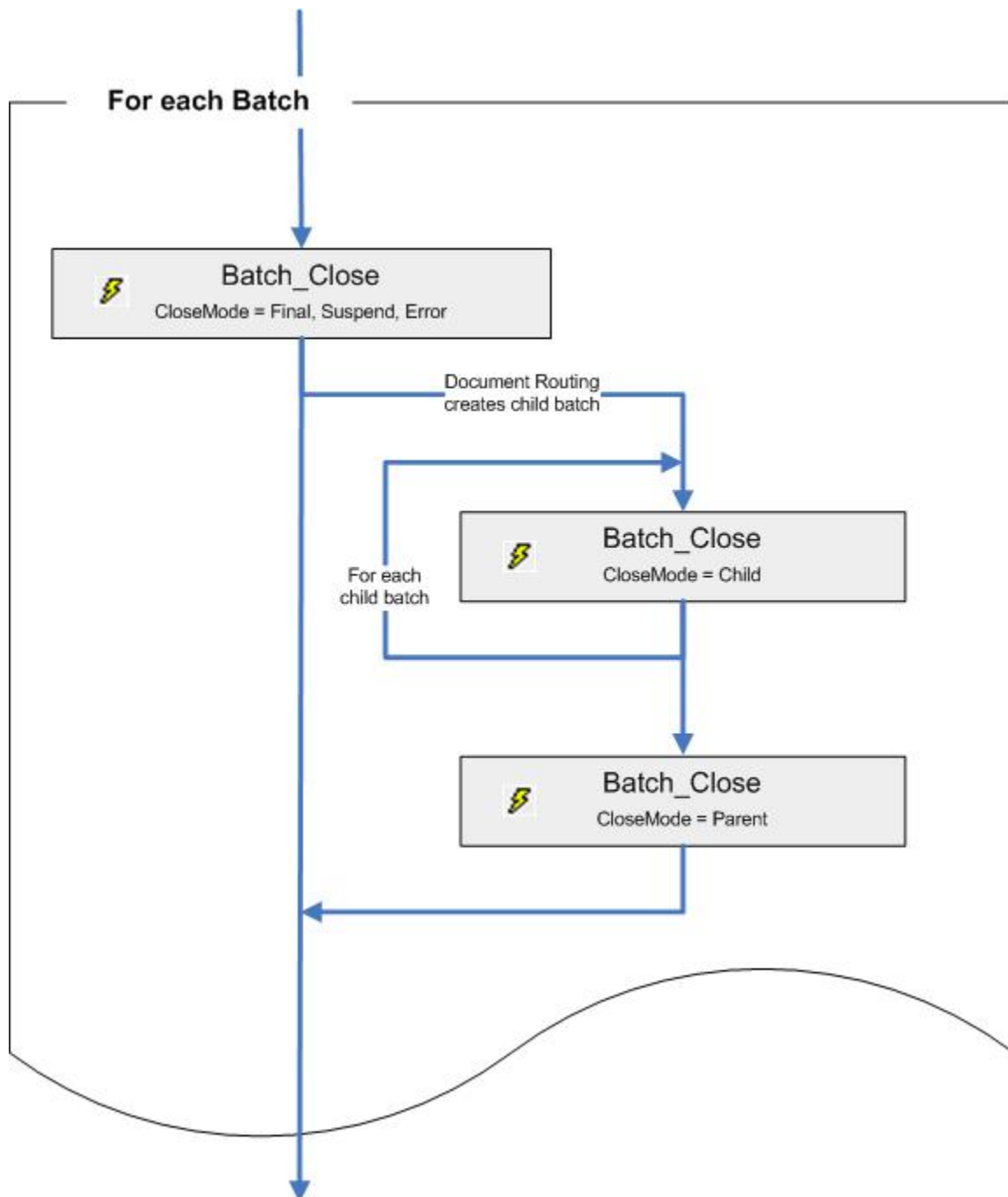
---

**Note** The `Batch_Close` event is fired to suspend, reject, and close batch status. Document Routing is run for each of these closing modes.

---

Document Routing proceeds in the following sequence:

- After the `Batch_Close` event the root hierarchy is checked for XValues.
- All XValues are analyzed to determine the next action for the documents.
- Documents are routed to the child batch.
- A `Batch_Close` event for the child batch is run, after which the child batch is routed to the designated queue module.
- The parent batch is routed to the designated queue module. If the parent batch is empty or contains only empty folders because all documents have been rerouted, the parent batch is deleted.



**Document Routing batch event sequence**

## Moving Documents to a child Batch

After the `Batch_Close` event, if the XValue with the key `KTM_DOCUMENTROUTING` in the document exists, its value defines a string placeholder that represents the child batch. Each unique string placeholder implicitly creates a child batch where the documents are moved. Documents that share the same value are routed to the same child batch.

Documents can be marked interactively per `Button_Click` event or at any point before or in the `Batch_Close` event. Afterwards, Document Routing is run. If the key is set to an empty string or is missing altogether, the document remains in the parent batch.

If the name of the child batch should be set explicitly by script, you need to define another XValue with the key `KTM_DOCUMENTROUTING_BATCHNAME_<Placeholder>` on root XFolder level with the value that is used as batch name. For example, the key `KTM_DOCUMENTROUTING_BATCHNAME_BATCH1` with the value "ErrorBatch" creates a child batch with that batch name. If the given batch name already exists in the Kofax Capture batch list, it is automatically made unique.

If the batch name is not set explicitly, the name of the child batch is derived from the name of the parent batch.

### Example. Routing a Rejected Document out of the Parent Batch to a Child Batch

```
Private Sub Batch_Close(ByVal pXRootFolder As CASCADELib.CscXFolder, ByVal CloseMode As
    CASCADELib.CscBatchCloseMode)
    If CloseMode = CscBatchCloseError Then
        Dim i As Long
        For i = 0 To pXRootFolder.GetTotalDocumentCount - 1
            Dim oXDocInfo As CASCADELib.CscXDocInfo
            Set oXDocInfo = pXRootFolder.GetDocInfoByGlobalIndex(i)
            If IsDocumentRejected(oXDocInfo) Then
                oXDocInfo.XValues.Set("KTM_DOCUMENTROUTING", "ErrorBatch")
            End If
        Next
    End If
End Sub

Private Function IsDocumentRejected(oXDocInfo As CASCADELib.CscXDocInfo) As Boolean
    IsDocumentRejected = False
    If oXDocInfo.XValues.ItemExists("AC_REJECTED_DOCUMENT") Then
        IsDocumentRejected = True
    Else
        Dim nPageIndex As Long
        For nPageIndex = 0 To oXDocInfo.PageCount - 1
            If oXDocInfo.XValues.ItemExists("AC_REJECTED_PAGE" & CStr(nPageIndex + 1)) Then
                IsDocumentRejected = True
            End If
        Next
    End If
End Function
```

## Assigning a Batch to Modules in the Queue

At the `Batch_Close` event, the XValue key `KTM_DOCUMENTROUTING_QUEUE_<NewBatchName>` is analyzed and the next queue process for a batch (the parent batch or the child batch) is assigned. The `<BatchName>` must be a unique identifier created for each child batch (designated as "Batch1", "Batch2" and so on). The designation `THISBATCH` means that the documents remain in the parent batch.

The value specified for the module must be the module ID or the alias of the process in the queue where you want the batch to be sent. For example, the key `KTM_DOCUMENTROUTING_QUEUE_BATCH1` with the value `KTM.Correction` means the next process for the batch is Kofax Transformation Modules Correction. For each child batch, a corresponding XValue (that is, a set unique identifier) assigns the queue module for processing.

---

**Note** If no module value is specified for a child batch, it is assigned the next process after the current module. If the module ID is invalid or does not exist in the current queue, the batch is sent to Quality Control.

---

The module names and their corresponding aliases for assigning batches to Kofax Capture and Kofax Transformation Modules queues are listed in the following table. For all custom modules not listed in the table use their module IDs to assign batches to queues.

**Table 8.1. Kofax Module Names**

Module	Legacy ModuleID	Alias ModuleID
	Kofax Capture	
KC Scan	Scan.exe	KC.Scan
KC Release	Release.exe	KC.Release
KCNS	ACIRSA.exe	KC.KCNS
KC Recognition Server	fp.exe	KC.Recognition
KC Validation	index.exe	KC.Validation
KC OCR Full Text	kfxpdf.exe	KC.PDF
KC PDF Generator	ocr.exe	KC.OCR
KC Quality Control	qc.exe	KC.QC
KC Verification	verify.exe	KC.Verification
	Kofax Transformation Modules	
KTM Server	LCI.Mailroom	KTM.Server
KTM Server 2	LCI.MailroomInst2	KTM.Server2
Kofax.DocumentReview	Kofax.DocumentReview	KTM.DocumentReview
KTM Correction	Kofax.Correction	KTM.Correction
KTM Validation step N	LCI.Validation<N>	KTM.Validation<N>
KTM KB Learning Server	LCI.KBModule	KTM.KBLearningServer
KTM Verification	Kofax.Verification	KTM.Verification

**Example. Route the Current Batch from Server to the Validation Instance where the Batch is Invalid**

```
Private Sub Batch_Close(ByVal pXRootFolder As CASCADELib.CscXFolder, ByVal CloseMode As
    CASCADELib.CscBatchCloseMode)
    Dim i As Long
    If CloseMode = CASCADELib.CscBatchCloseFinal And Project.ScriptExecutionMode =
        CscScriptModeServer Then
        For i = 0 To Project.ValidationInstances - 1
            If pXRootFolder.ValidForInstance(Project.FolderValidationInstance, i) = False Then
                pXRootFolder.XValues.Set("KTM_DOCUMENTROUTING_QUEUE_THISBATCH", "KTM.Validation" &
                    CStr(i + 1))
            End If
        Next
    End If
End Sub
```

## Batch Editing

Batch Editing is available in document review and validation. The following table shows the availability of the different batch editing operations for document review, validation (distinguished for rich client and thin client). If a batch editing operation is available for both processing steps this can be distinguished by the property `Project.ScriptExecutionMode`.

**Table 8.2. Page operations**

User interaction and corresponding events	Document Review	Validation	Thin Client Va
<a href="#">Adding Page</a>	✓	✓	
<a href="#">Deleting Page</a>	✓	✓	
<a href="#">Rotating Page</a>	✓	✓	

**Table 8.3. Document operations**

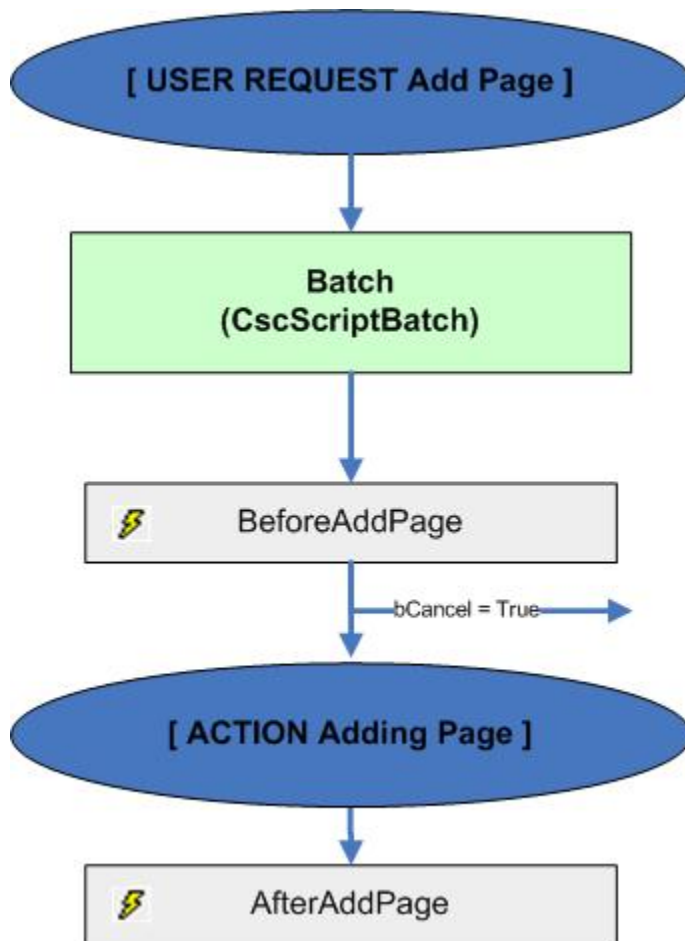
User interaction and corresponding events	Document Review	Validation	Thin Client Va
<a href="#">Adding Document</a>	✓	✓	
<a href="#">Before Overriding Document Problem</a>	✓	✗	
<a href="#">Before Restoring Document Problem</a>	✓	✗	
<a href="#">Changing Class</a>	✓	✗	
<a href="#">Confirm Class</a>	✓	✗	
<a href="#">Copying Document</a>	✓	✓	
<a href="#">Creating Document</a>	✓	✓	
<a href="#">Deleting Document</a>	✓	✓	
<a href="#">Merging Document</a>	✓	✓	
<a href="#">Moving Document</a>	✓	✓	
<a href="#">Splitting Document</a>	✓	✓	

**Table 8.4. Folder operations**

User interaction and corresponding events	Document Review	Validation	Thin Client Va
<a href="#">Before Overriding Folder Problem</a>	✓	✗	
<a href="#">Before Restoring Folder Problem</a>	✓	✗	
<a href="#">Creating Folder</a>	✗	✓	
<a href="#">Deleting Folder</a>	✗	✓	
<a href="#">Merging Folder</a>	✗	✓	
<a href="#">Moving Folder</a>	✗	✓	
<a href="#">Splitting Folder</a>	✗	✓	

## Adding Page

This diagram shows the sequence of events initiated by the user request to add a page. In the `BeforeAddPage` event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Add Page of the Batch object in project script**

#### **Example. Cancel adding pages to single page documents**

```

Private Sub Batch_BeforeAddPage(pXDocInfo As CASCADELib.CscXDocInfo, bCancel As Boolean)
    If pXDocInfo.XDocument.ExtractionClass = "SinglePageForm" Then
        MsgBox "You cannot add pages to a single page form. Please create a new document."
        bCancel = True
    End If
End Sub

```

#### **Example. Copy the added page to an additional destination**

Reference Microsoft Scripting Runtime used.

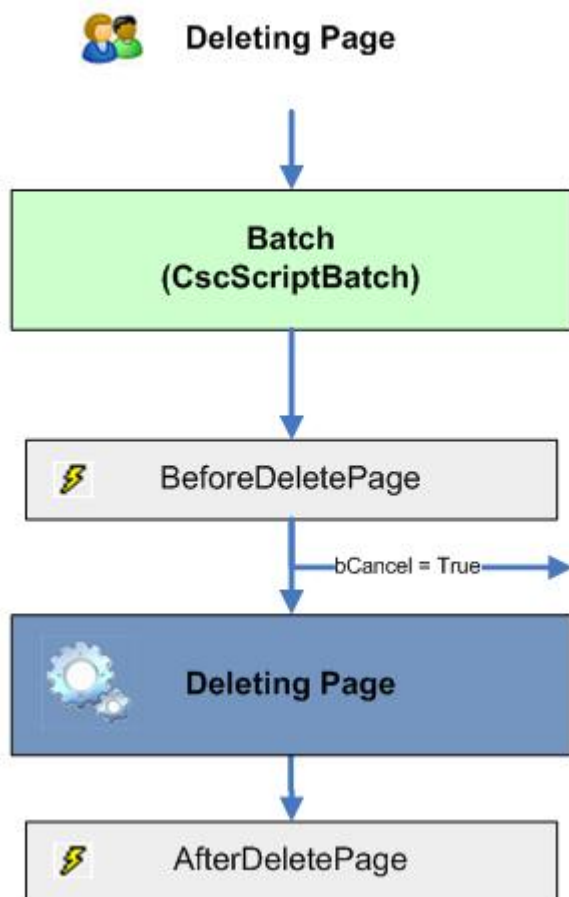
```

Private Sub Batch_AfterAddPage(pXDocInfo As CASCADELib.CscXDocInfo, ByVal PageIndex As Long)
    Dim FSO As New FileSystemObject
    FSO.CopyFile(pXDocInfo.Filename(PageIndex), "c:\temp", True)
End Sub

```

## **Deleting Page**

This diagram shows the sequence of events initiated by the user request to delete a page. In the BeforeDeletePage event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Delete Page of the Batch object in project script**

#### **Example. Disallow deleting of first pages**

The deletion of the first page is canceled. Only pages after the first page can be deleted.

```

Private Sub Batch_BeforeDeletePage(pXDocInfo As CASCADELib.CscXDocInfo, ByVal PageIndex As
Long, bCancel As Boolean)
    If PageIndex = 0 Then
        bCancel = True
    End If
End Sub

```

#### **Example. Count deleted pages**

Counts all deleted pages in a document and saves this information to a specific Key/Value pair from which it can be retrieved at any time.

```

Private Sub Batch_AfterDeletePage(pXDocInfo As CASCADELib.CscXDocInfo, ByVal DeletedPageIndex
As Long)
    Dim nCount As Long
    If pXDocInfo.XValues.ItemExists("DeletedPageCount") Then
        nCount = CLng(pXDocInfo.XValues("DeletedPageCount"))
        pXDocInfo.XValues.Set("DeletedPageCount", CStr(nCount + 1))
    Else
        pXDocInfo.XValues.Add("DeletedPageCount", "1", True)
    End If
End Sub

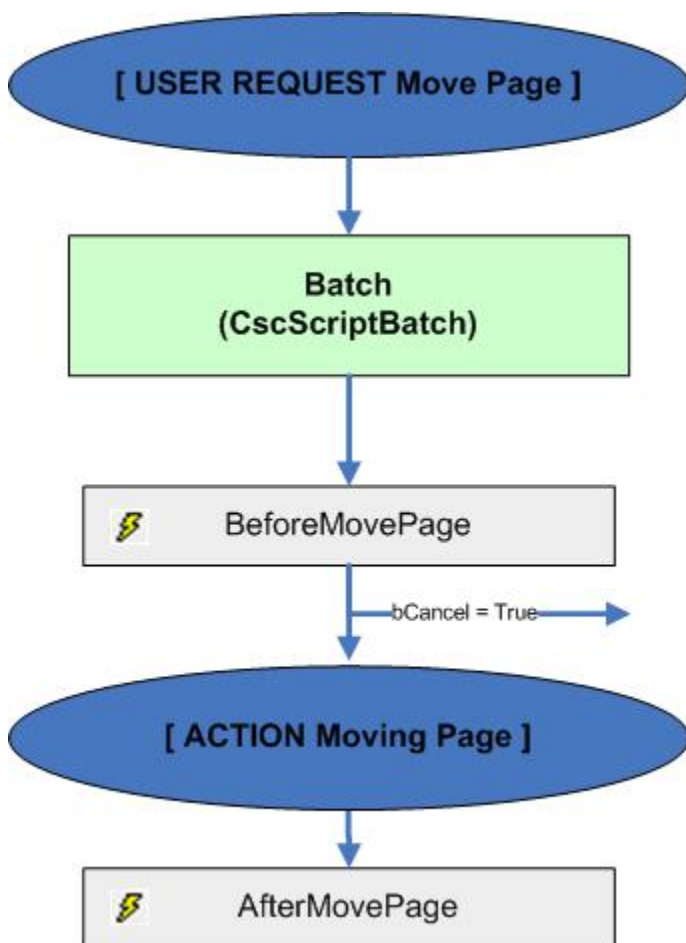
```



End Sub

## Moving Page

This diagram shows the sequence of events initiated by the user request to move a page. In the BeforeMovePage event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Move Page of the Batch object in project script**

### Example. Allow page movement only within the same document

```

Private Sub Batch_BeforeMovePage(pTargetXDocInfo As CASCADELib.CscXDocInfo, ByVal
    TargetPageIndex As Long, pSrcXDocInfo As CASCADELib.CscXDocInfo, ByVal SrcPageIndex As Long,
    bCancel As Boolean)
    If pTargetXDocInfo.Filename <> pSrcXDocInfo.Filename Then
        bCancel = True
    End If
End Sub
End Sub
  
```

### Example. Confirming first document field again because of move operation

Let the user again confirm the first field of the document(s) that were affected by the page move operation.

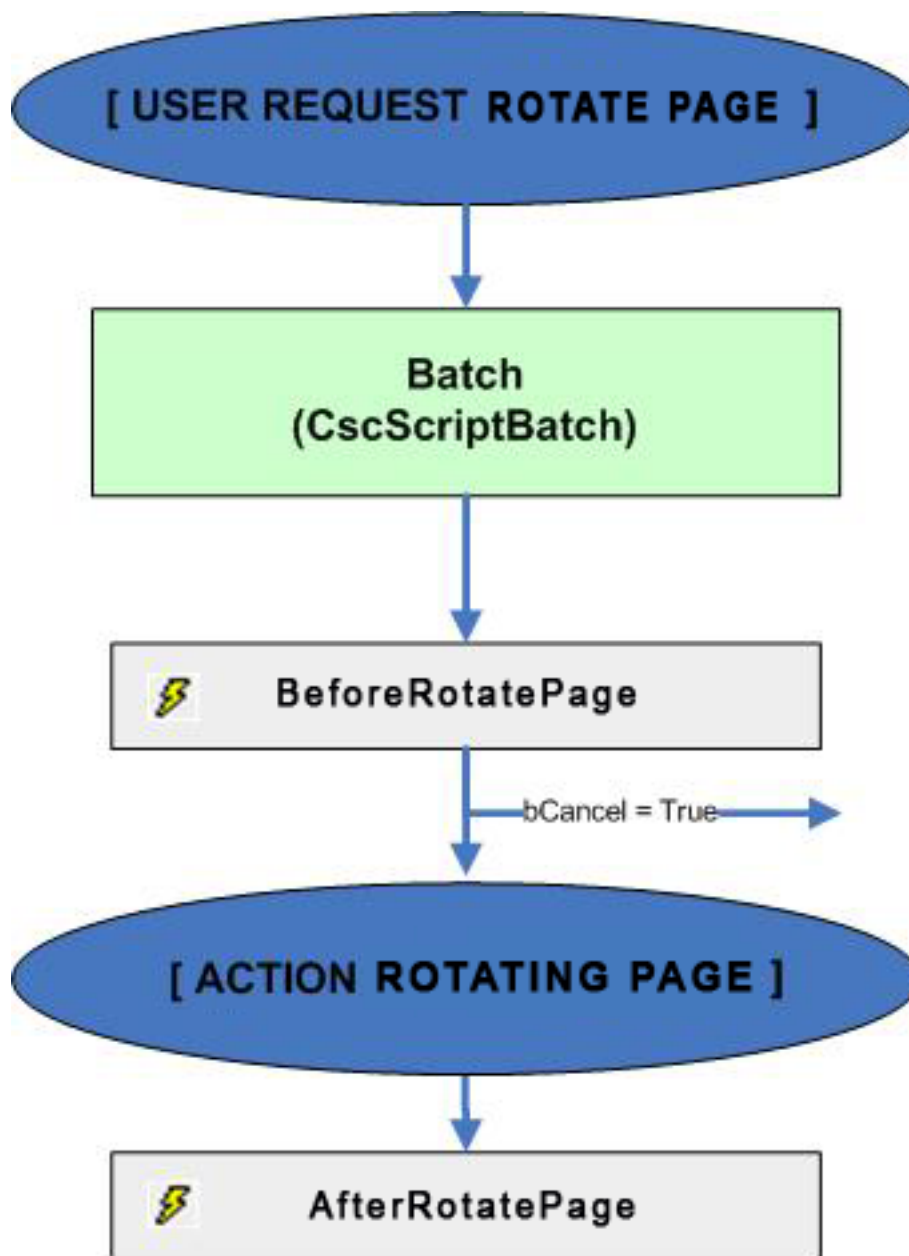
```

Private Sub Batch_AfterMovePage(pTargetXDocInfo As CASCADELib.CscXDocInfo, ByVal
TargetPageIndex As Long, pSrcXDocInfo As CASCADELib.CscXDocInfo, ByVal SrcPageIndex As Long)
' pSrcXDocInfo might be nothing if the last page was removed
If Not pSrcXDocInfo Is Nothing Then
' field count might be 0 if the documents are still unclassified
If pSrcXDocInfo.XDocument.Fields.Count > 0 Then
pSrcXDocInfo.XDocument.Fields(0).Valid = False
pSrcXDocInfo.XDocument.Fields(0).ErrorDescription = "Please confirm ths field again
because a page was removed."
End If
End If
' field count might be 0 if the document is still unclassified
If pTargetXDocInfo.XDocument.Fields.Count > 0 Then
pTargetXDocInfo.XDocument.Fields(0).Valid = False
pTargetXDocInfo.XDocument.Fields(0).ErrorDescription = "Please confirm this field again
because a page was inserted."
End If
End Sub

```

## Rotating Page

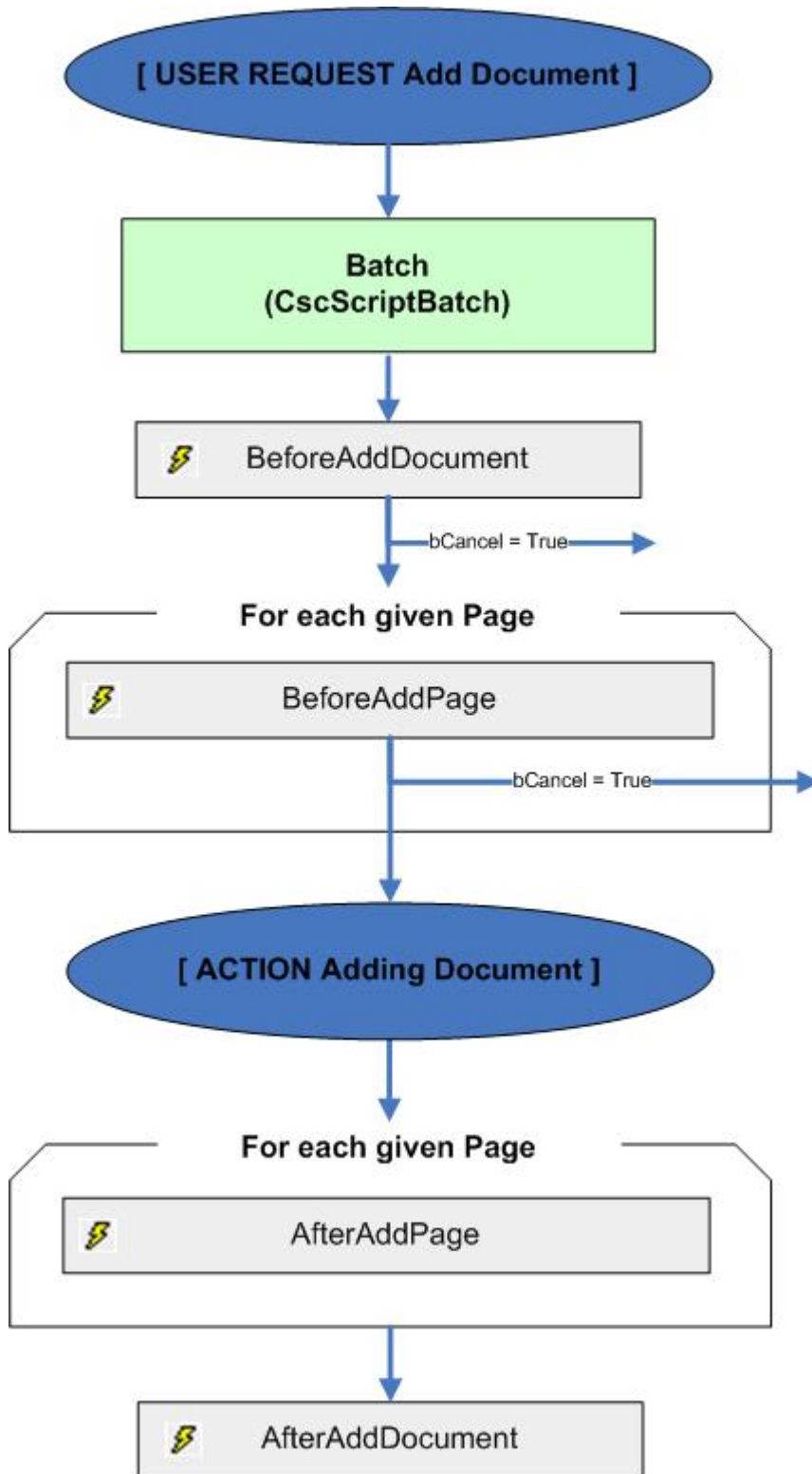
This diagram shows the sequence of events initiated by the user request to rotate a page. In the BeforeRotatePage event, the user interaction can be canceled. To verify if the opening module is the current module use the property Project.ScriptExecutionMode.



Event sequence during the user request Rotate Page of the Batch object in project script

## Adding Document

This diagram shows the sequence of events initiated by the user request to add a document. In the BeforeAddDocument event and in each BeforeAddPage event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Add Document of the Batch object in project script**

**Example. Assigning default classification result to all added documents**

```
Private Sub Batch_BeforeAddDocument(sClassName As String, bCancel As Boolean)
```

```
sClassName = Project.ClassByID(Project.DefaultClassID).Name
End Sub
```

### Example. Cancel adding pages to documents of a specific class

```
Private Sub Batch_BeforeAddPage(pXDocInfo As CASCADELib.CscXDocInfo, bCancel As Boolean)
    If pXDocInfo.ExtractionClass = "XYZ" Then
        bCancel = True
    End If
End Sub
```

### Example. Copy added page to a different location

Reference Microsoft Scripting Runtime used.

```
Private Sub Batch_AfterAddPage(pXDocInfo As CASCADELib.CscXDocInfo, ByVal PageIndex As Long)
    Dim FSO As New FileSystemObject
    FSO.CopyFile(pXDocInfo.Filename(PageIndex), "c:\temp", True)
End Sub
```

### Example. Increment the added document count of the parent folder by one

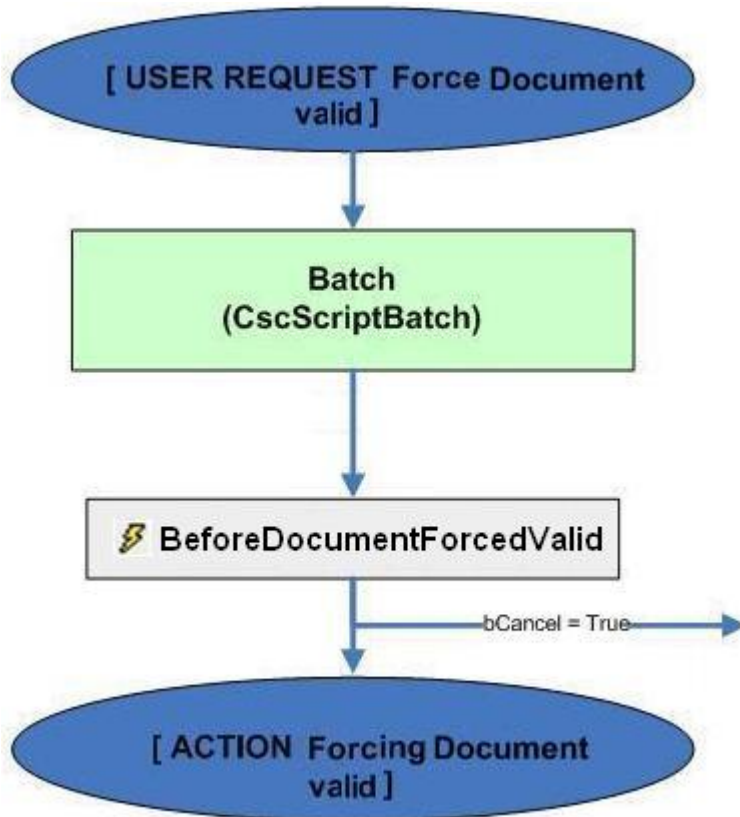
Foldering must be activated, and the parent folder object must contain a field with the name "AddedDocumentCount".

```
Private Sub Batch_AfterAddDocument(pXDocInfo As CASCADELib.CscXDocInfo)
    Dim nDocCount As Long
    If pXDocInfo.ParentFolder.Fields.Exists("AddedDocumentCount") Then
        If Len(pXDocInfo.ParentFolder.Fields.ItemByName("AddedDocumentCount").Text) > 0 Then
            nDocCount = CLng(pXDocInfo.ParentFolder.Fields.ItemByName("AddedDocumentCount").Text)

        Else
            nDocCount = 0
        End If
        pXDocInfo.ParentFolder.Fields.ItemByName("AddedDocumentCount").Text = nDocCount + 1
    End If
End Sub
```

## Before Overriding Document Problem

This diagram shows the sequence of events initiated by the user request to override a document problem. In the BeforeOverrideDocumentProblem event, the user request can be canceled. This event is only available for the Document Review module, additionally you can check the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Force Document Valid of the Batch object in project script**

#### Example. Handling reason text for overriding problem

Ensures that the operator provides a reason for overriding the problem.

```

Private Sub Batch_BeforeOverrideDocumentProblem(ByRef pXDocInfo As CASCADELib.CscXDocInfo,
ByRef bCancel As Boolean)

    Dim sReason As String

    sReason = InputBox("Please enter a reason for overriding the problem")

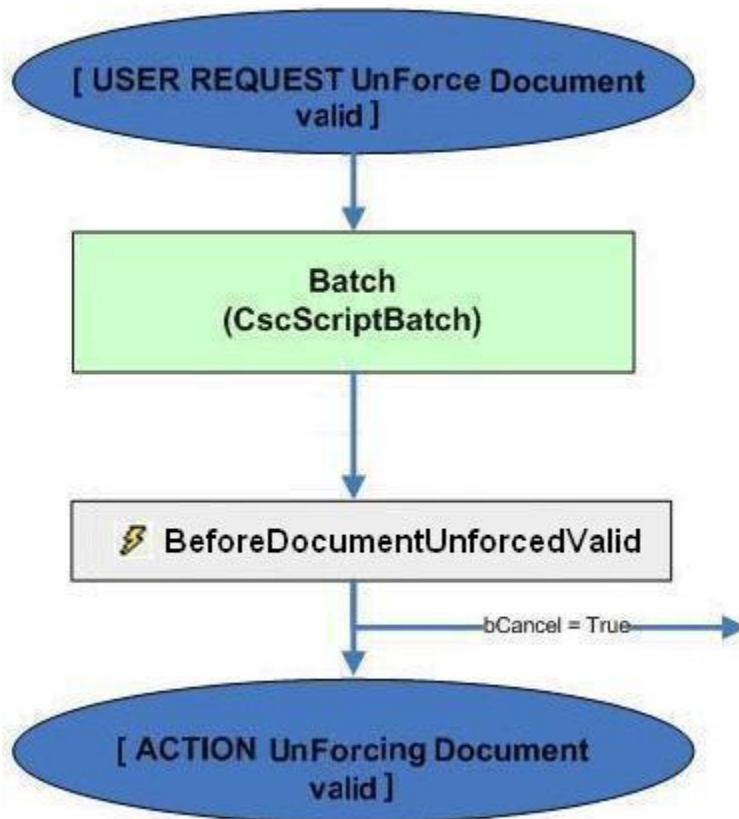
    If sReason = "" Then
        MsgBox "You cannot override a problem without entering a reason", vbCritical
        bCancel = True
    Else
        pXDocInfo.XDocument.DocumentReviewForceValidReason = sReason
    End If

End Sub

```

### Before Restoring Document Problem

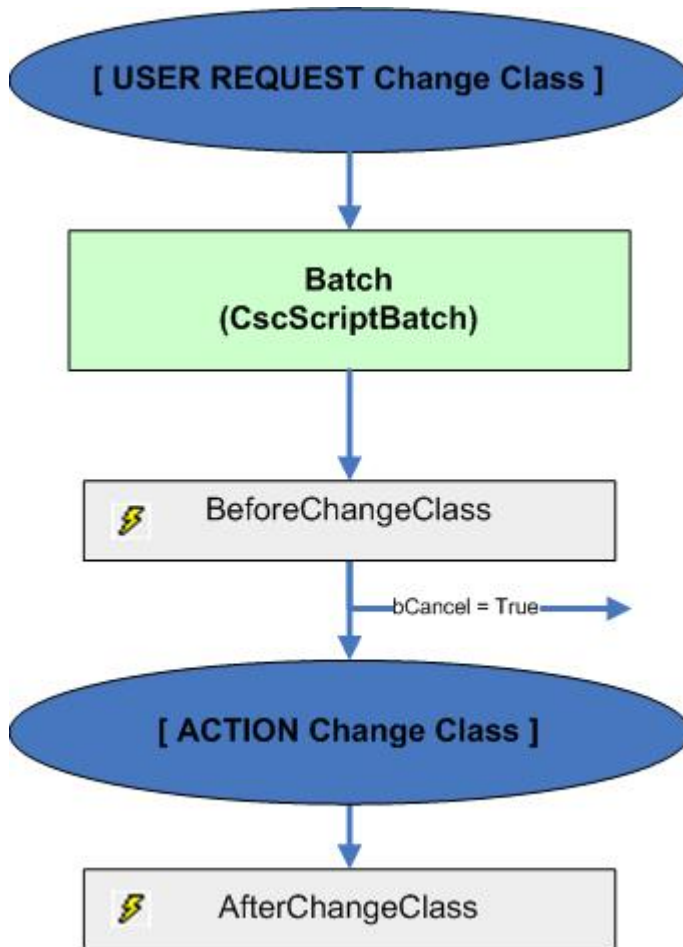
This diagram shows the sequence of events initiated by the user request to clear the “forced valid” status of a document and restore the problem. In the `BeforeRestoreDocumentProblem` event, the user interaction can be canceled. This event is only available for the Document Review module, additionally you can check the property `Project.ScriptExecutionMode`.



Event sequence during the user request Clear Forced Valid of the Batch object in project script

## Changing Class

This diagram shows the sequence of events initiated by the user request to change the classification result of a document. In the `BeforeChangeClass` event, the user interaction can be canceled. This event is only available for the Document Review module, additionally you can check the property `Project.ScriptExecutionMode`.

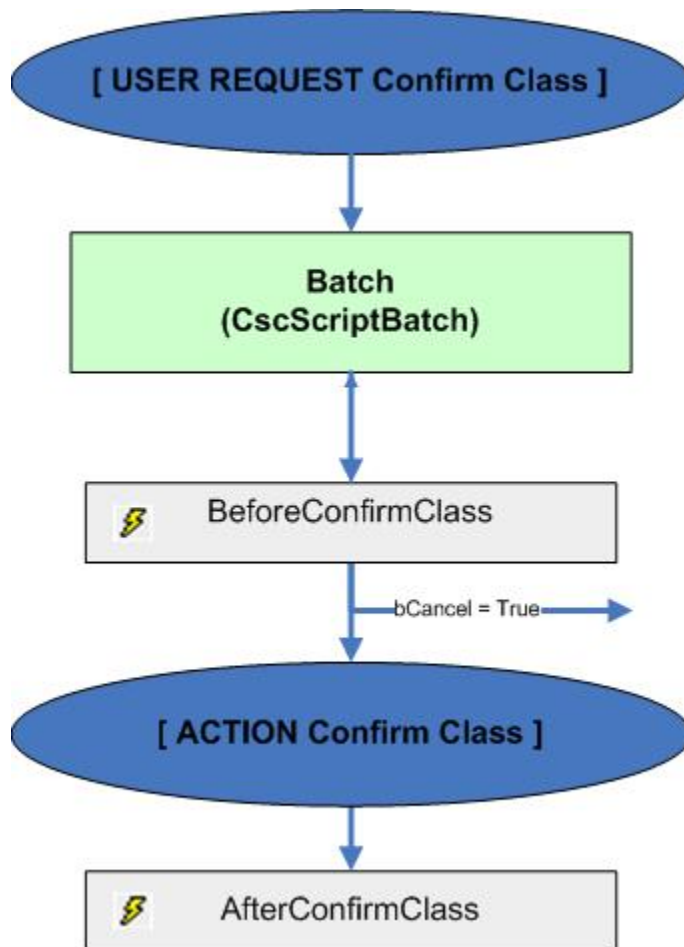


**Event sequence during the user request Change Class of the Batch object in project script**

### Confirming Class

This diagram shows the sequence of events initiated by the user request to confirm a class. In the `BeforeConfirmClass` event, the user interaction can be canceled. This event is only available for the Document Review module, additionally you can check the property `Project.ScriptExecutionMode`.

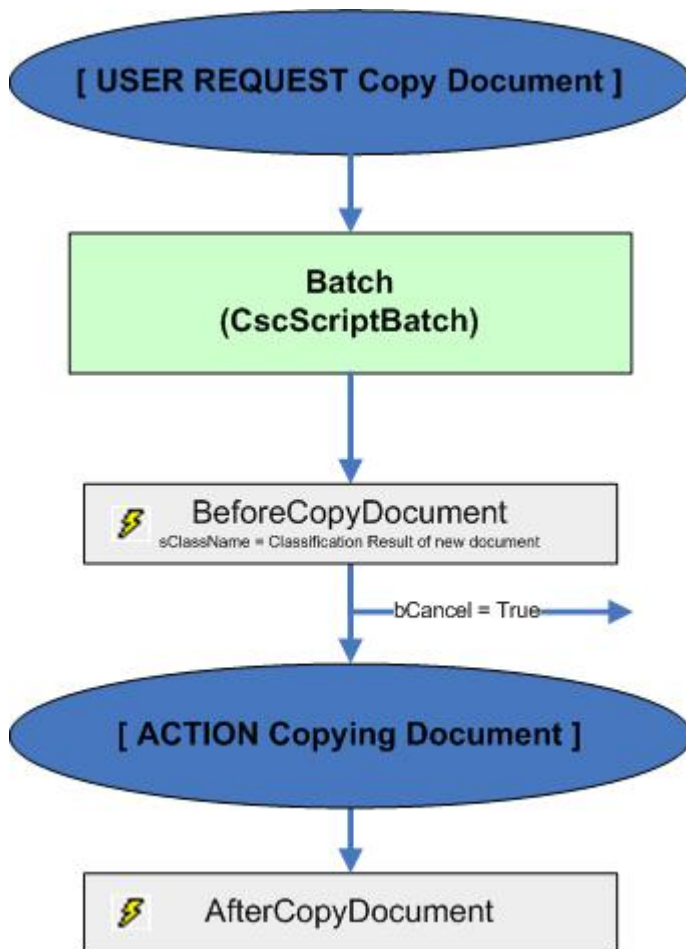




**Event sequence during the user request Confirm Class of the Batch object in project script**

## Copying Document

This diagram shows the sequence of events initiated by the user request to copy a document. In the `BeforeCopyDocument` event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Copy Document of the Batch object in project script**

#### **Example. Condition Copying of a document**

Copy the document only if the copied document is classified to a specific class ("ClassA"), and assign that class to the copied document. If the copied document has another classification result, cancel the copy action.

```

Private Sub Batch_BeforeCopyDocument(pSrcXDocInfo As CASCADELib.CscXDocInfo, sClassName As
String, bCancel As Boolean)
    If pSrcXDocInfo.XDocument.ExtractionClass = "ClassA" Then
        sClassName = pSrcXDocInfo.XDocument.ExtractionClass
    Else
        bCancel = True
    End If
End Sub

```

#### **Example. Copy field results to copied document**

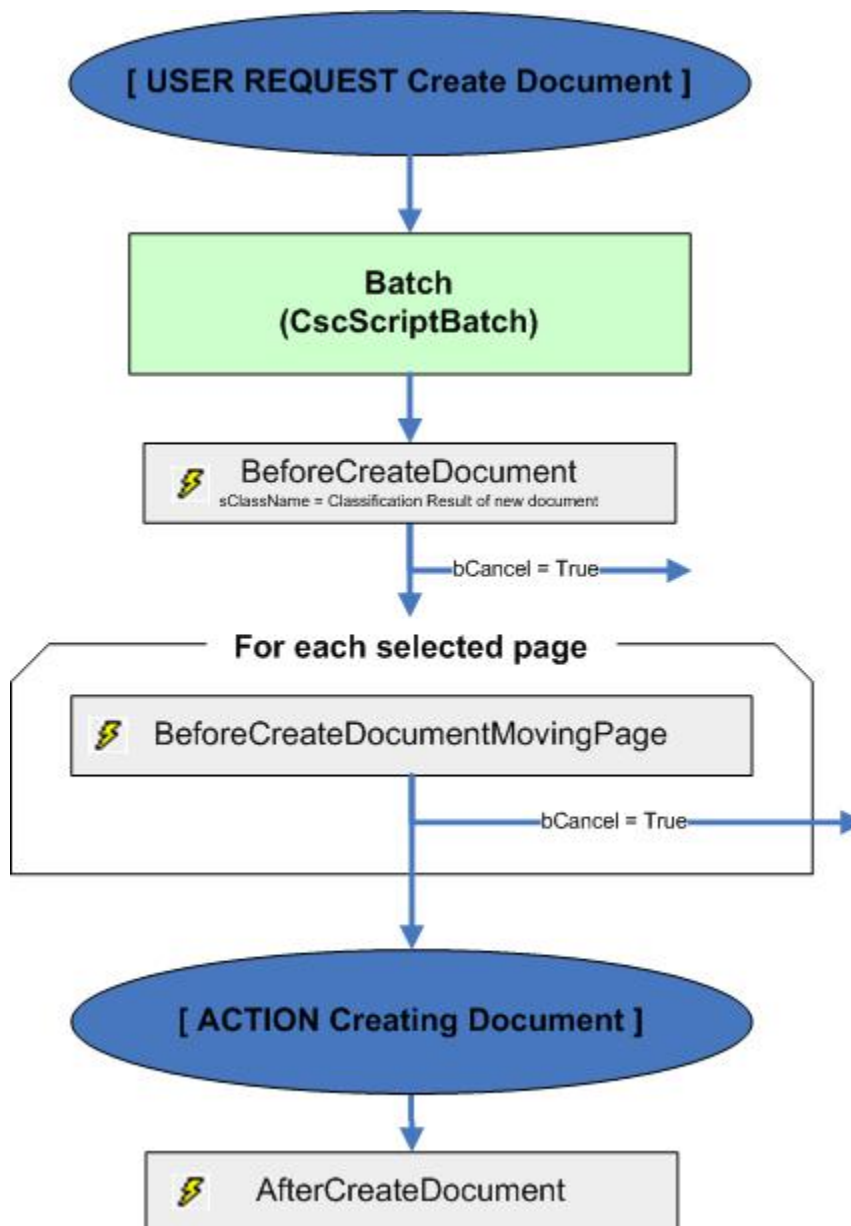
```

Private Sub Batch_AfterAddDocument(pSrcXDocInfo As CASCADELib.CscXDocInfo, pNewXDocInfo As
CASCADELib.CscXDocInfo)
    Dim n As Long
    For n = 0 To pSrcXDocInfo.XDocument.Fields.Count - 1
        pNewXDocInfo.XDocument.Fields(n).InitFromField(pSrcXDocInfo.XDocument.Fields(n))
    Next n
End Sub

```

## Creating Document

This diagram shows the sequence of events initiated by the user request to create a document. In the `BeforeCreateDocument` event and in each `BeforeCreateDocumentMovingPage` event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



Event sequence during the user request Create Document of the Batch object in project script

### Example. Assign classification results to created documents

Initialize all newly-created documents with the class name of the source document if it is classified to a specific class ("ClassA"). Otherwise, use the default classification result defined in the project as class name for the created document.

```

Private Sub Batch_BeforeCreateDocument(pSrcXDocInfo As CASCADELib.CscXDocInfo, sClassName As
String, bCancel As Boolean)
    If pSrcXDocInfo.XDocument.ExtractionClass = "ClassA" Then
        sClassName = pSrcXDocInfo.XDocument.ExtractionClass
    Else
        sClassName = Project.ClassByID(Project.DefaultClassID).Name
    End If
End Sub

```

### Example. Condition document creation by classification result

Cancel document creation if the user has selected pages of a document that should not be changed.

```

Private Sub Batch_BeforeCreateDocumentMovingPage(pSrcXDocInfo As CASCADELib.CscXDocInfo, ByVal
SrcPageIndex As Long, _
pNewXDocInfo As CASCADELib.CscXDocInfo, ByVal NewPageIndex As Long, bCancel As Boolean)
    If pSrcXDocInfo.XDocument.ExtractionClass = "Important" Then
        MsgBox("The document creation is canceled because documents with the classification
result 'Important' cannot be changed.")
        bCancel = True
    End If
End Sub

```

### Example. Reclassify all created documents to the defined default class

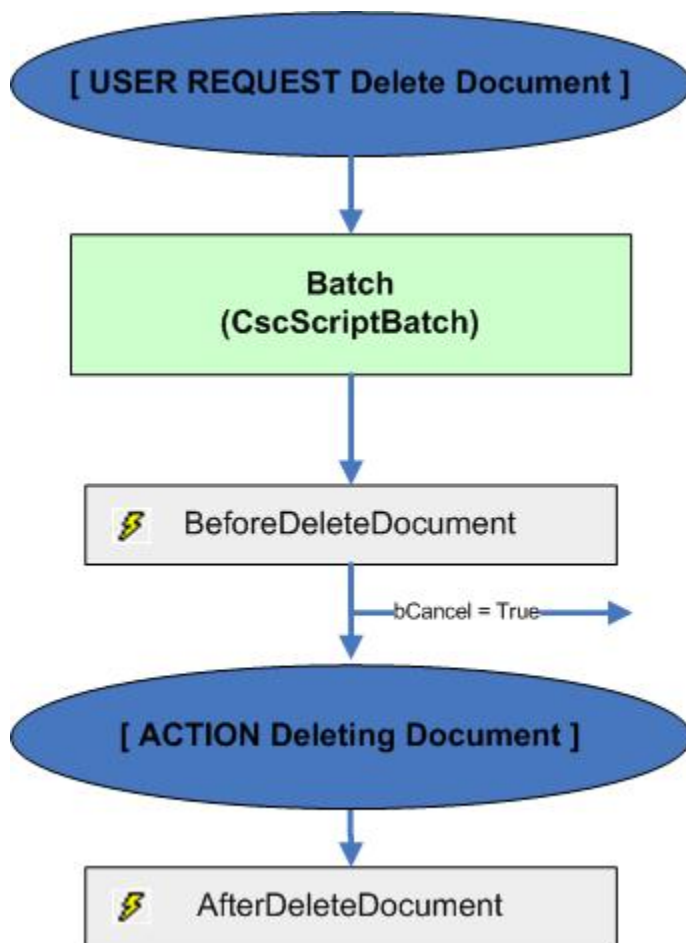
```

Private Sub Batch_AfterCreateDocument(pNewXDocInfo As CASCADELib.CscXDocInfo)
    pNewXDocInfo.XDocument.Reclassify(Project.ClassByID(Project.DefaultClassID).Name)
End Sub

```

## Deleting Document

This diagram shows the sequence of events initiated by the user request to delete a document. In the `BeforeDeleteDocument` event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Delete Document of the Batch object in project script**

**Example. Avoid the deletion of documents with a specific classification result**

```

' Project classification script
' ...
Private Sub Batch_BeforeDeleteDocument(pXDocInfo As CASCADELib.CscXDocInfo, bCancel As Boolean)
    If pXDocInfo.XDocument.ExtractionClass = "ClassA" Then
        bCancel = True
    End If
End Sub

```

**Example. Recalculates a total value from all other documents and assigns it to a folder field**

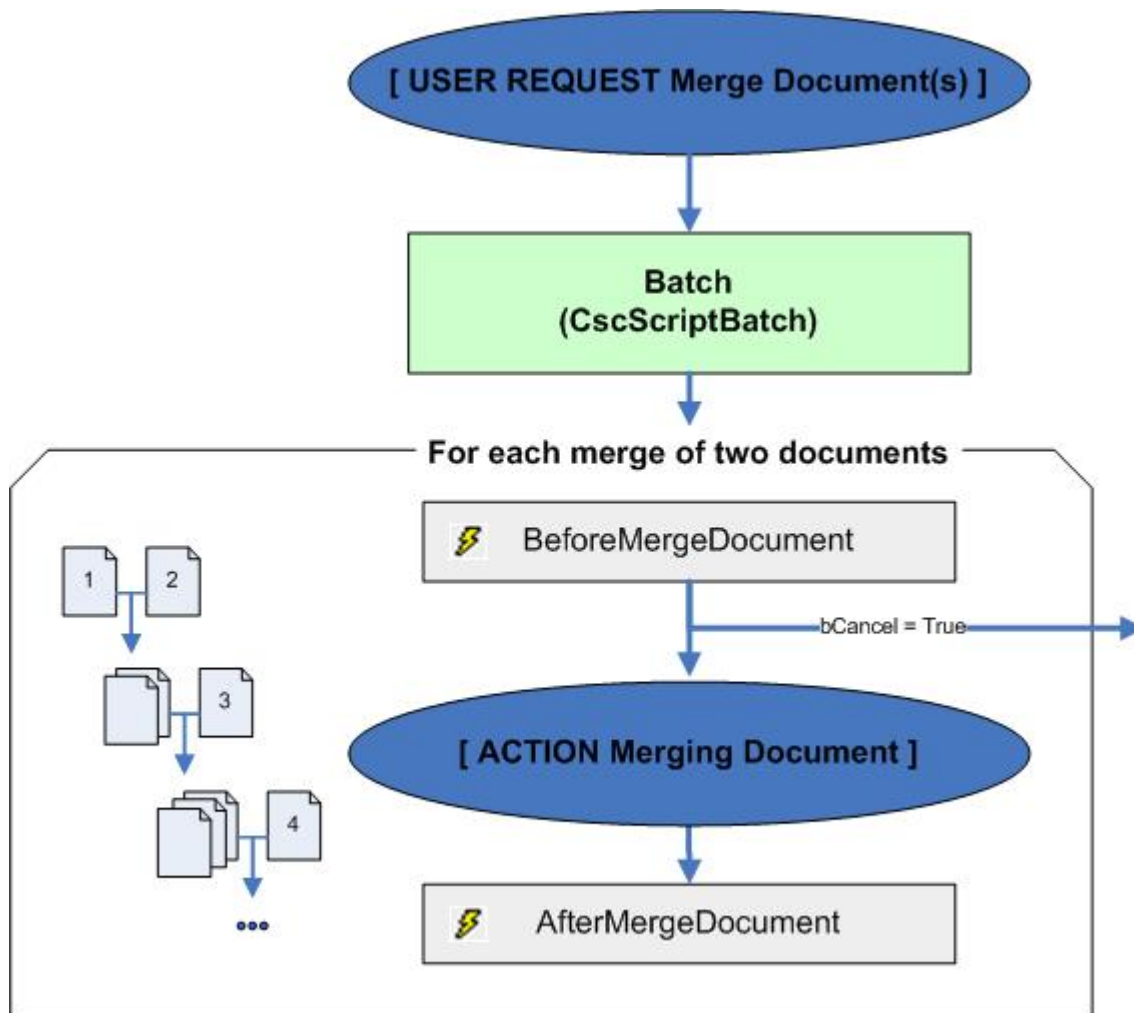
```

Private Sub Batch_AfterDeleteDocument(pParentXFolder As CASCADELib.CscXFolder, ByVal DeletedDocumentIndex As Long)
    Dim n As Long
    Dim nSum As Long
    Dim oXDocInfo As CASCADELib.CscXDocInfo
    For n = 0 To pParentXFolder.DocInfos.Count - 1
        oXDocInfo = pParentXFolder.DocInfos(n)
        If oXDocInfo.Fields.Exists("Total") Then
            nSum = nSum + CLng(oXDocInfo.Fields.ItemByName("Total").Value)
        End If
    Next n
    If pParentXFolder.Fields.Exists("Total") Then
        pParentXFolder.Fields.ItemByName("Total").Value = nSum
    End If
End Sub

```

## Merging Document

This diagram shows the sequence of events initiated by the user request to merge two or more documents into one document. Each merge of two documents is handled as one event sequence. This event sequence is then executed for each additional document that is merged. In the `BeforeMergeDocument` event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Merge Document of the Batch object in project script**

### Example. Sample BeforeMergeDocument

Only allow the merging of documents with the same classification result

```

' Project classification script
' ...
Private Sub Batch_BeforeMergeDocument(pFirstXDocInfo As CASCADELib.CscXDocInfo, pSecondXDocInfo
As CASCADELib.CscXDocInfo, bCancel As Boolean)
    If pFirstXDocInfo.XDocument.ExtractionClass <> pSecondXDocInfo.XDocument.ExtractionClass
    Then
        bCancel = True
    End If
End Sub
  
```

### Example. Confirm field again after merging document

Ask the user again to confirm the first field of a document again after the document was merged.

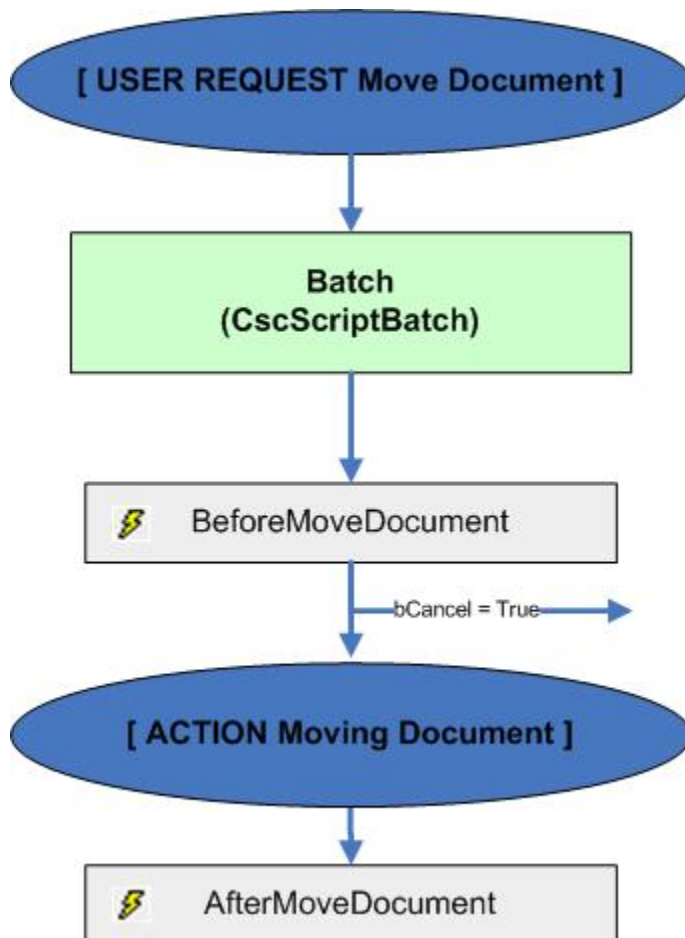
```

Private Sub Batch_AfterMergeDocument(pMergedXDocInfo As CASCADELib.CscXDocInfo, ByVal
MergedPageIndex As Long)
    pMergedXDocInfo.XDocument.Fields(0).Valid = False
    pMergedXDocInfo.XDocument.Fields(0).ErrorDescription = "Please confirm this field again
after the document was merged."
    End If
End Sub

```

## Moving Document

This diagram shows the sequence of events initiated by the user request to move a document. In the `BeforeMoveDocument` event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Move Document of the Batch object in project script**

**Example. Allowing document move operations only inside the same folder**

```

' Project classification script
' ...
Private Sub Batch_BeforeMoveDocument(pXDocInfo As CASCADELib.CscXDocInfo, pTargetXFolder As
CASCADELib.CscXFolder, ByVal TargetDocIndex As Long, pSrcXFolder As CASCADELib.CscXFolder,
ByVal SrcDocIndex As Long, bCancel As Boolean)
    If pTargetXFolder.GlobalIndex <> pSrcXFolder.GlobalIndex Then
        bCancel = True
    End If
End Sub

```

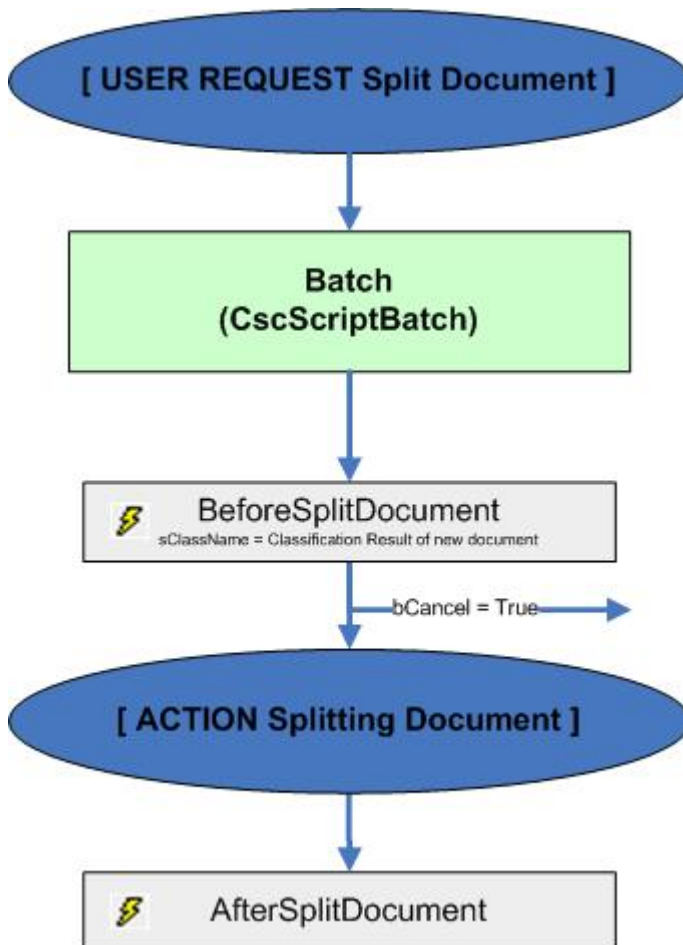
**Example. Use field information of first document for folder**

Writes the value of a specific field to a folder field if the first document has changed during moving operations in a folder

```
Private Sub Batch_AfterMoveDocument(pXDocInfo As CASCADELib.CscXDocInfo, pTargetXFolder As
CASCADELib.CscXFolder, ByVal TargetDocIndex As Long, pSrcXFolder As CASCADELib.CscXFolder,
ByVal SrcDocIndex As Long)
    If TargetDocIndex = 0 Then
        If pTargetXFolder.Fields.Exists("FirstDocInfo") And
pXDocInfo.Fields.Exists("FirstDocInfo") Then
            pTargetXFolder.Fields.ItemByName("FirstDocInfo").Value =
pXDocInfo.Fields.ItemByName("FirstDocInfo").Value
        End If
    End If
End Sub
```

**Splitting Document**

This diagram shows the sequence of events initiated by the user request to split a document. In the BeforeSplitDocument event, the user interaction can be canceled. To verify if the opening module is the current module use the property Project.ScriptExecutionMode.



**Event sequence during the user request Split Document of the Batch object in project script**

**Example. Condition splitting of document on classification result**

Avoids splitting documents with a specific classification result. If splitting is executed, the second document receives the same classification result as the original one.



```

' Project classification script
' ...
Private Sub Batch_BeforeSplitDocument(pSrcXDocInfo As CASCADELib.CscXDocInfo, ByVal PageIndex
As Long, sClassname As String, bCancel As Boolean)
    If pSrcXDocInfo.XDocument.ExtractionClass = "ClassA" Then
        bCancel = True
    Else
        sClassName = pSrcXDocInfo.XDocument.ExtractionClass
    End If
End Sub
End Sub

```

### Example. Transfer field data to split document

Copies all values from the first original document to the splitted second one.

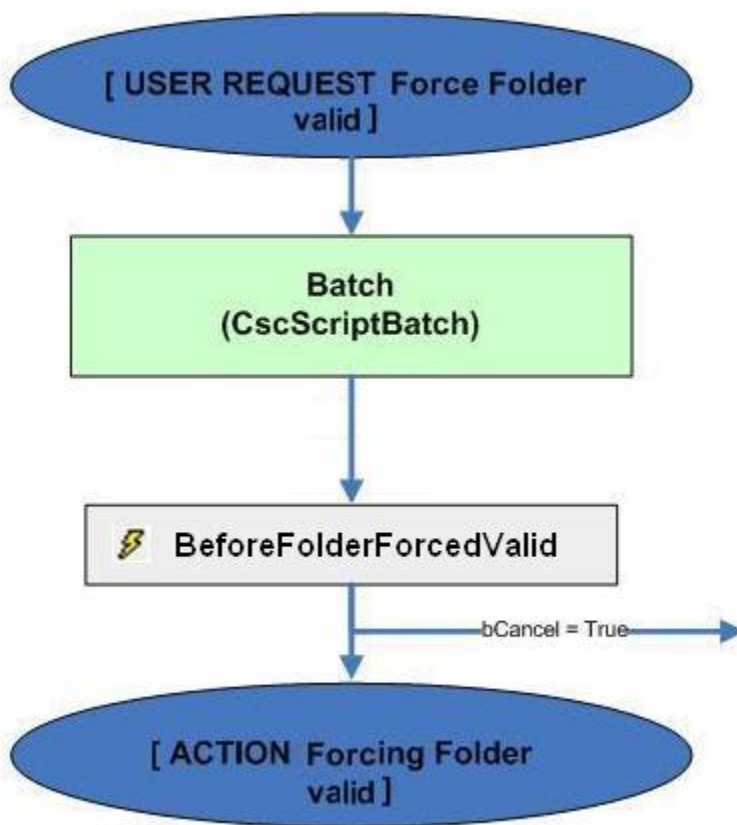
```

Private Sub Batch_AfterSplitDocument(pFirstXDocInfo As CASCADELib.CscXDocInfo, pSecondXDocInfo
As CASCADELib.CscXDocInfo)
    Dim n As Long
    For n = 0 To pSrcXDocInfo.XDocument.Fields.Count - 1
        pSecondXDocInfo.XDocument.Fields(n).InitFromField(pFirstXDocInfo.XDocument.Fields(n))
    Next n
End Sub

```

## Before Overriding Folder Problem

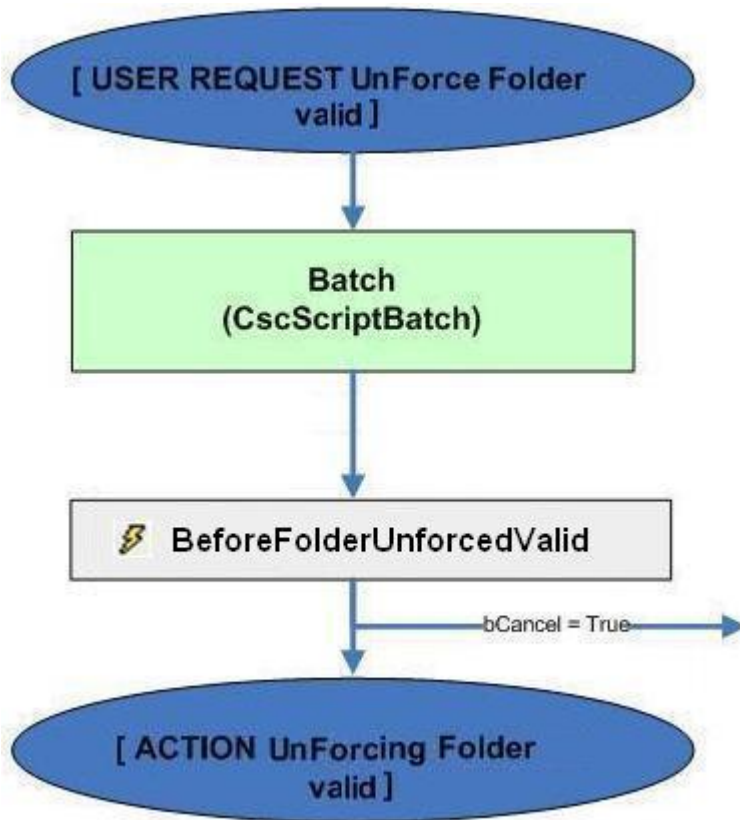
This diagram shows the sequence of events initiated by the user request to force a problem batch into a valid state. In the BeforeFolderForcedValid event, the user interaction can be canceled. This event is only available for the Document Review module, additionally you can check the property `Project.ScriptExecutionMode`.



Event sequence during the user request Force Batch Valid of the Batch object in project script

## Before Restoring Folder Problem

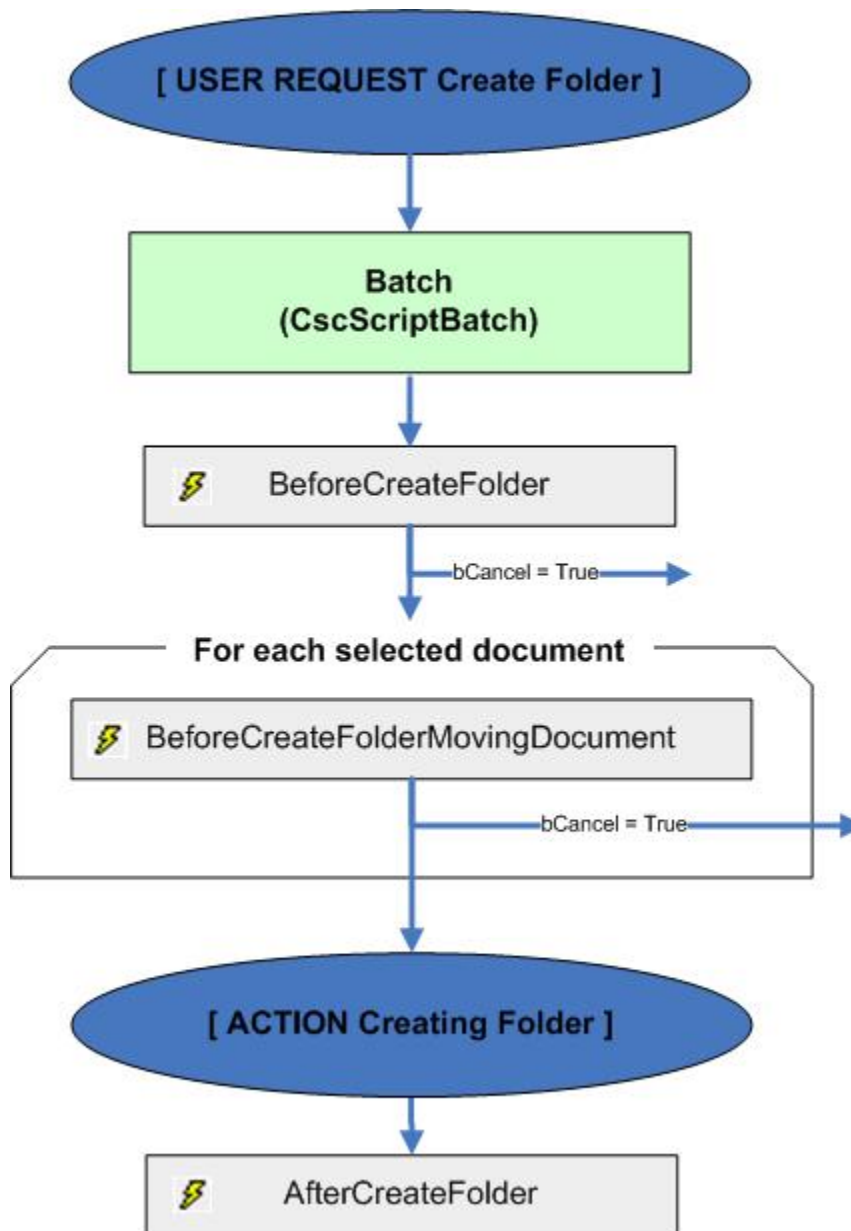
This diagram shows the sequence of events initiated by the user request to clear the “forced valid” status of the batch and restore the problem. In the `BeforeFolderUnforcedValid` event, the user interaction can be canceled. This event is only available for the Document Review module, additionally you can check the property `Project.ScriptExecutionMode`.



Event sequence during the user request Clear Forced Valid of the Batch object in project script

## Creating Folder

This diagram shows the sequence of events initiated by the user request to create a folder from the selected documents or from an empty one. In the `BeforeCreateFolder` event and in each `BeforeCreateFolderMovingDocument` event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Create Folder of the Batch object in project script**

#### **Example. Condition folder creation on document count**

Allows folder creation with only one or two documents on level one.

```

' Project classification script
' ...
Private Sub Batch_BeforeCreateFolder(pNewXFolder As CASCADELib.CscXFolder, ByVal
    TotalDocumentCount As Long, bCancel As Boolean)
    If TotalDocumentCount > 2 Or pNewXFolder.Level <> 1 Then
        bCancel = True
    End If
End Sub

```

#### **Example. Disallow moving documents of a specific classification result**

```

Private Sub Batch_BeforeCreateFolderMovingDocument(pXDocInfo As CASCADELib.CscXDocInfo,
    pSrcXFolder As CASCADELib.CscXFolder, _

```

```

    ByVal SrcDocIndex As Long, pTargetXFolder As CASCADELib.CscXFolder, ByVal TargetDocIndex As
Long, bCancel As Boolean)
    If pXDocInfo.XDocument.ExtractionClass = "ClassA" Then
        bCancel = True
    End If
End Sub

```

### Example. Calculate folder field from documents

Calculates the sum of a specific field from all documents and assigns the value a field of the new folder.

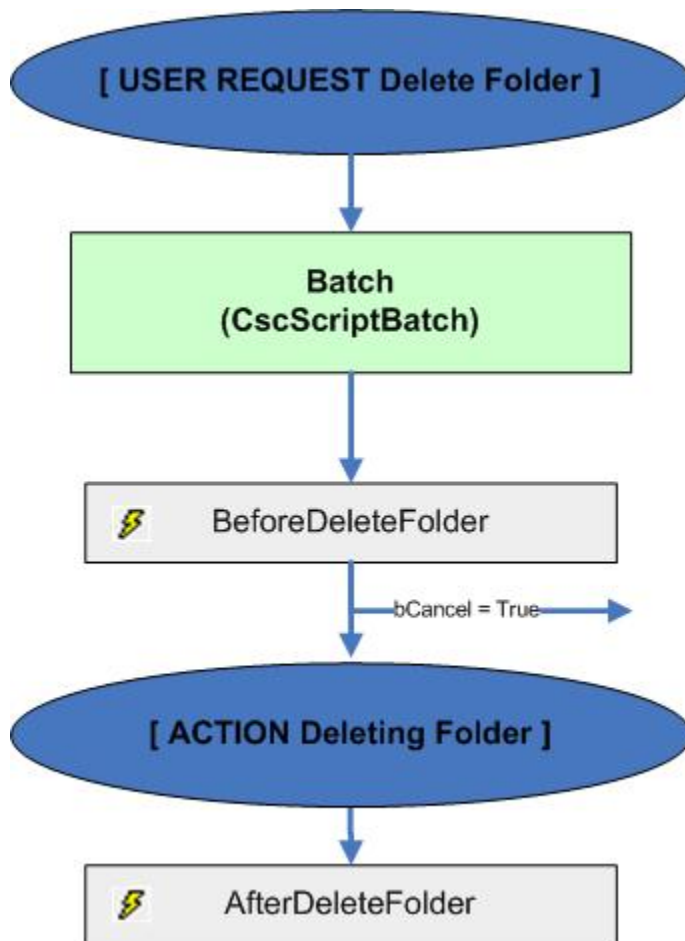
```

Private Sub Batch_AfterCreateFolder(pNewXFolder As CASCADELib.CscXFolder)
    Dim n As Long
    Dim oXDocInfo As CASCADELib.CscXDocInfo
    Dim Sum As Long
    Sum = 0
    For n = 0 To pNewXFolder.DocInfos.Count - 1
        If oXDocInfo.XDocument.Fields.Exists("Total") Then
            Sum = Sum + CLng(oXDocInfo.XDocument.Fields.ItemByName("Total").Value)
        End If
    Next n
    If pNewXFolder.Fields.Exists("Total") Then
        pNewXFolder.Fields.ItemByName("Total").Value = Sum
    End If
End Sub

```

## Deleting Folder

This diagram shows the sequence of events initiated by the user request to delete a folder. In the BeforeDeleteFolder event, the user interaction can be canceled. To verify if the opening module is the current module please use the property Project.ScriptExecutionMode.



**Event sequence during the user request Delete Folder of the Batch object in project script**

#### **Example. Control deletion of folder dependent of level**

Avoids the deletion of folders that are at level one. Only folders on a deeper level can be deleted.

```

Private Sub Batch_BeforeDeleteFolder(pXFolder As CASCADELib.CscXFolder, bCancel As Boolean)
    If pXFolder.Level < 2 Then
        bCancel = True
    End If
End Sub

```

#### **Example. Recalculate a folder field from field of sub folders**

Calculates the sum of a folder field of the child folders from which one folder was deleted and assigns the new calculated value to the parent folder.

```

Private Sub Batch_AfterDeleteFolder(pParentXFolder As CASCADELib.CscXFolder, ByVal DeletedFolderIndex As Long)
    Dim n As Long
    Dim nSum As Long
    Dim oXFolder As CASCADELib.CscXFolder
    For n = 0 To pParentXFolder.Folders.Count - 1
        oXFolder = pParentXFolder.Folders(n)
        If oXFolder.Fields.Exists("Total") Then
            nSum = nSum + CLng(oXFolder.Fields.ItemByName("Total").Value)
        End If
    Next n
    pParentXFolder.Fields.ItemByName("Total").Value = nSum
End Sub

```

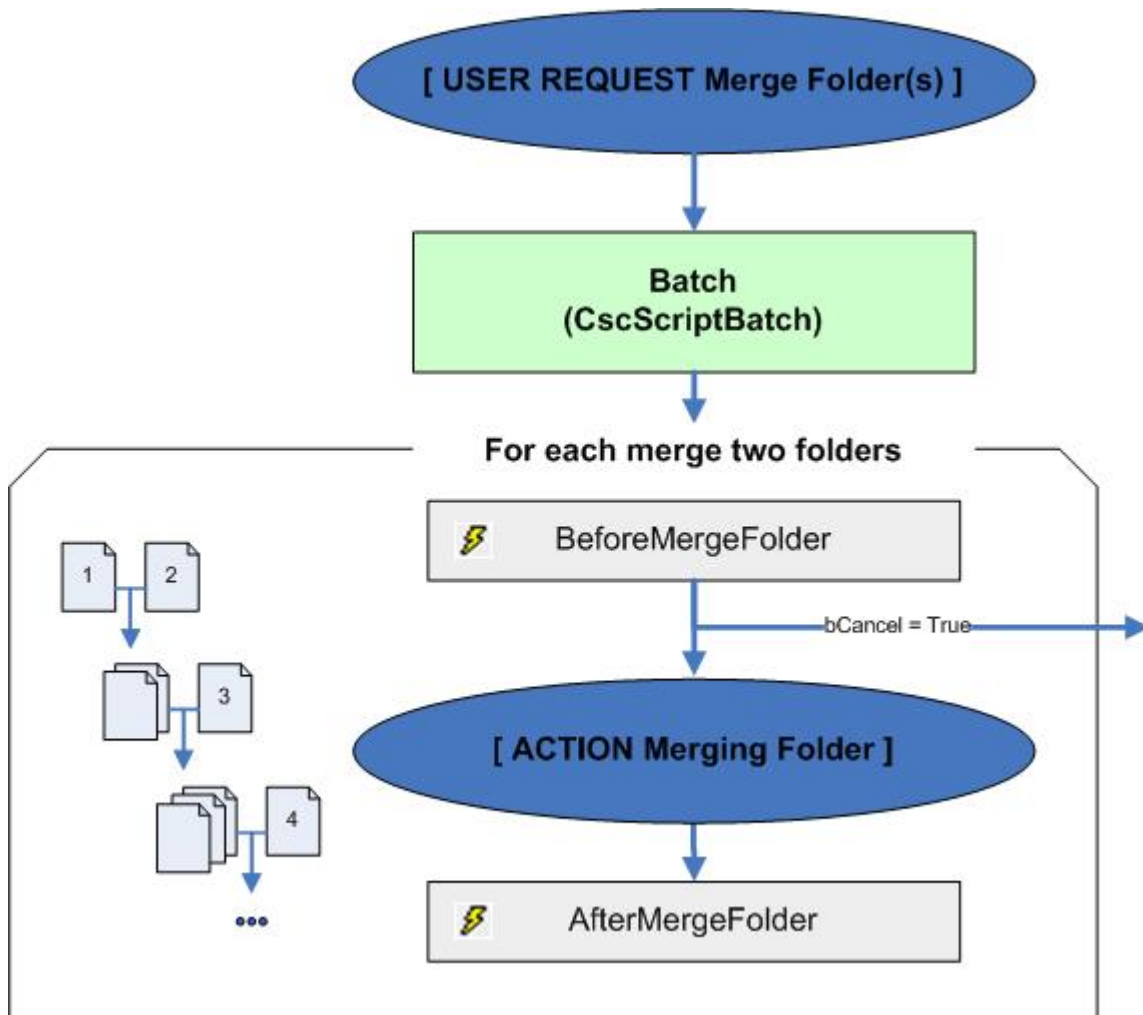
```

Next n
If pParentXFolder.Fields.Exists("Total") Then
    pParentXFolder.Fields.ItemByName("Total").Value = nSum
End If
End Sub

```

## Merging Folder

This diagram shows the sequence of events initiated by the user request to merge two or more folders into one folder. Each merge of two folders is handled as one event sequence. This event sequence is then executed for each additional folder that is merged. In the `BeforeMergeFolder` event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



**Event sequence during the user request Merge Folder of the Batch object in project script**

**Example. Allows folder merging only on level two and higher**

```

Private Sub Batch_BeforeMergeFolder(pFirstXFolder As CASCADELib.CscXFolder, pSecondXFolder As
    CASCADELib.CscXFolder, bCancel As Boolean)
    If pFirstXFolder.Level < 2 Then
        bCancel = True
    End If
End Sub

```

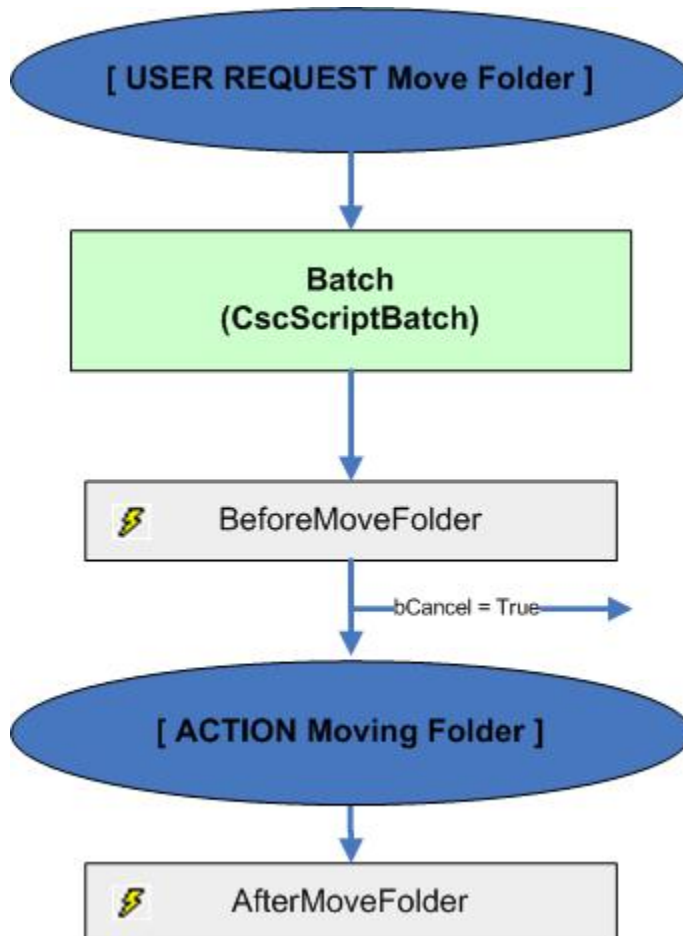
```
End Sub
```

### Example. Recalculates a total field in folder over all documents

```
Private Sub Batch_AfterMergeFolder(pMergedXFolder As CASCADELib.CscXDocInfo, ByVal
MergedDocumentIndex As Long)
    Dim n As Long
    Dim nSum As Long
    Dim oXDocInfo As CASCADELib.CscXDocInfo
    For n = 0 To pMergedXFolder.DocInfos.Count - 1
        oXDocInfo = pMergedXFolder.DocInfos(n)
        If oXDocInfo.Fields.Exists("Total") Then
            nSum = nSum + CLng(oXDocInfo.Fields.ItemByName("Total").Value)
        End If
    Next n
    If pMergedXFolder.Fields.Exists("Total") Then
        pMergedXFolder.Fields.ItemByName("Total").Value = nSum
    End If
End Sub
```

## Moving Folder

This diagram shows the sequence of events initiated by the user request to move a folder. In the BeforeMoveFolder event, the user interaction can be canceled. To verify if the opening module is the current module please use the property `Project.ScriptExecutionMode`.



Event sequence during the user request Move Folder of the Batch object in project script

**Example. Sample BeforeMoveFolder**

Only folders on levels higher than one can be moved.

```
' Project classification script
' ...
Private Sub Batch_BeforeMoveFolder(pXFolder As CASCADELib.CscXFolder, pTargetParentXFolder
As CASCADELib.CscXFolder, ByVal TargetFolderIndex As Long, pSrcParentXFolder As
CASCADELib.CscXFolder, ByVal SrcFolderIndex As Long, bCancel As Boolean)
    If pXFolder.Level < 2 Then
        bCancel = True
    End If
End Sub
```

**Example. Sample AfterMoveFolder**

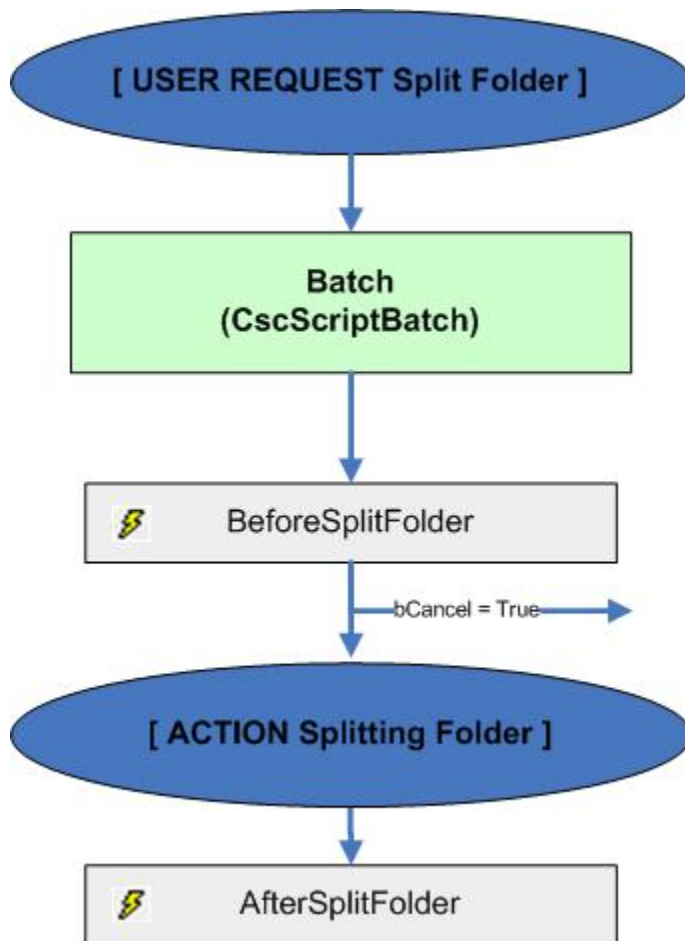
Recalculates a sum folder field for both parent folders from the contained child folders if the moved folder is moved to a different parent folder

```
Private Sub Batch_AfterMoveFolder(pXFolder As CASCADELib.CscXFolder, pTargetParentXFolder
As CASCADELib.CscXFolder, ByVal TargetFolderIndex As Long, pSrcParentXFolder As
CASCADELib.CscXFolder, ByVal SrcFolderIndex As Long)
    If pSrcParentXFolder.GlobalIndex <> pTargetParentXFolder.GlobalIndex Then
        Dim n As Long
        Dim nSum As Long
        Dim oXFolder As CASCADELib.CscXFolder
        ' update the source parent folder of the moved folder
        For n = 0 To pSrcParentXFolder.Folders.Count - 1
            oXFolder = pSrcParentXFolder.Folders(n)
            If oXFolder.Fields.Exists("Total") Then
                nSum = nSum + CLng(oXFolder.Fields.ItemByName("Total").Value)
            End If
        Next n
        If pSrcParentXFolder.Fields.Exists("Total") Then
            pSrcParentXFolder.Fields.ItemByName("Total").Value = nSum
        End If
        ' update the new parent of the moved folder
        For n = 0 To pTargetParentXFolder.Folders.Count - 1
            oXFolder = pTargetParentXFolder.Folders(n)
            If oXFolder.Fields.Exists("Total") Then
                nSum = nSum + CLng(oXFolder.Fields.ItemByName("Total").Value)
            End If
        Next n
        If pTargetParentXFolder.Fields.Exists("Total") Then
            pTargetParentXFolder.Fields.ItemByName("Total").Value = nSum
        End If
    End If
End Sub
```

**Splitting Folder**

This diagram shows the sequence of events initiated by the user request to split a folder. In the BeforeSplitFolder event, the user interaction can be canceled. To verify if the opening module is the current module please use the property Project.ScriptExecutionMode.





**Event sequence during the user request Split Folder of the Batch object in project script**

#### **Example. Sample BeforeSplitFolder**

The splitting of a folder is executed only for folders that are on level one.

```

' Project classification script
' ...
Private Sub Batch_BeforeSplitFolder(pSrcXFolder As CASCADELib.CscXFolder, ByVal DocumentIndex
  As Long, bCancel As Boolean)
  If pSrcXFolder.Level > 1 Then
    bCancel = True
  End If
End Sub

```

#### **Example. Sample AfterSplitFolder**

Recalculates a sum from a folder field from all child folders for the first original folder and the new second one.

```

Private Sub Batch_AfterSplitFolder(pFirstXFolder As CASCADELib.CscXFolder, pSecondXFolder As
  CASCADELib.CscXFolder)
  Dim n As Long
  Dim nSum As Long
  Dim oXFolder As CASCADELib.CscXFolder
  ' update the first folder from its child folders
  For n = 0 To pFirstXFolder.Folders.Count - 1
    oXFolder = pFirstXFolder.Folders(n)
    If oXFolder.Fields.Exists("Total") Then
      nSum = nSum + CLng(oXFolder.Fields.ItemByName("Total").Value)
    End If
  Next n
  pFirstXFolder.Fields.ItemByName("Total").Value = nSum
End Sub

```

```

        End If
    Next n
    If pFirstXFolder.Fields.Exists("Total") Then
        pFirstXFolder.Fields.ItemByName("Total").Value = nSum
    End If
    ' update the second folder from its child folders
    For n = 0 To pSecondXFolder.Folders.Count - 1
        oXFolder = pSecondXFolder.Folders(n)
        If oXFolder.Fields.Exists("Total") Then
            nSum = nSum + CLng(oXFolder.Fields.ItemByName("Total").Value)
        End If
    Next n
    If pSecondXFolder.Fields.Exists("Total") Then
        pSecondXFolder.Fields.ItemByName("Total").Value = nSum    End If
End Sub

```

## Line Item Matching Locator

The Line Item Matching Locator provides an additional `Match` function that is used via script to match an existing table result against the purchase order database. This is performed for documents that contain an existing table result (such as electronic documents), or to check manually indexed table line items during validation.

To access the Line Item Matching Locator's `Match` functionality via script define the Line Item Matching Locator in the project first. If processed documents have already a table result and the line items from such a table have to be matched against the purchase order database during Server processing, the locator definition can be added to a separate class that is not used, to avoid the execution of the locator during extraction. Alternatively, (for example, if you have only a small number of documents) you can define the locator in the same class the document will be classified to, but you have to protect existing table results. Set the `Preserve` flag to `TRUE` to protect the table results from being overwritten.

You can use the `Match` function of a Line Item Matching Locator that is used to extract table data during Server processing in Validation if the used table model contains all global columns that are listed below.

### Table Model

If you use the match functionality in script the used table model must have the following global columns, otherwise an error is returned and the executed script is stopped:

- Order Number
- Description
- Quantity
- Unit price
- Total price
- PO Item Number

Matching returns the number of successfully matched lines. Matched line item results are written to the Order Number and PO Item Number columns to show the data that has been found in the purchase order database. If matching fails, the PO Item Number displays -1. You can configure the `Match` function to overwrite existing results.

When the row is matched, all cells of the row are valid, with the following exceptions:

- If the unit price is missing or scaled, the column Unit Price is invalid.
- If the total price is scaled, the column Total Price is invalid.
- If only the total price is found, only the Total Price is set to valid; the other cells of the row are invalid.

- If there are OCR errors or matching fails, the row is invalid.

**Match** (ByVal pXDoc As CASCADELib.CscXDocument, ByVal sTableFieldName As String, ByVal sPONumbers() As String, ByVal sVendorID As String, ByVal bOverwriteAll) As Long

### pXDoc

XDocument object that contains the table line items to match.

### sTableFieldName

Name of the table field that contains the line items to match.

### sPONumbers()

Purchase order numbers used for matching the line items; this version supports only one. The formatter setting for the purchase order number is applied to all items in the string array.

### sVendorID

Vendor ID used for matching the line items, this value is optional and is filled with an empty string.

### bOverwriteAll

Flag that determines if the cells in the table are overwritten by the Match function. If set to TRUE existing results are deleted and new results are entered. If set to FALSE only empty cells are filled with the matching results.

### Return: Long

Returns the number of matched line items.

The following method is used in both in the Validation and Server processing.

```
Private Function Match(pXDoc As CscLineItemMatchingLocLib.ICscXDocument, sClassName As String,
sLocatorName As String, sTableFieldName As String, sPONumber As String, sVendorID As String)
As Long
    Dim pClass As CASCADELib.CscClass
    Dim pLocatorDef As CASCADELib.CscLocatorDef

    ' get the locator definition to use for the matching
    Set pClass = Project.ClassByName(sClassName)
    Set pLocatorDef = pClass.Locators.ItemByName("LIMLoc")

    ' convert it to the LineItemMatchingLocator interface to call the match method
    Dim oIExtMethod As CASCADELib.ICscExtractionMethod
    Set oIExtMethod = pLocatorDef.LocatorMethod
    Dim oILIMLoc As CscLineItemMatchingLocLib.CscLineItemMatchingLocator
    Set oILIMLoc = oIExtMethod

    ' give the PONumber and/or the VendorID that should be used for matching
    Dim sPONumbers() As String
    ReDim sPONumbers(0)
    sPONumbers(0) = sPONumber
    Match = oILIMLoc.Match(pXDoc, sTableFieldName, sPONumbers, sVendorID, True)
    ' if 0 is returned nothing have been matched, table has not been changed.

End Function
```

## Using Match Function In Validation

Using the Match function in Validation allows you to get the matching information for an existing table field result that has been changed by the Validation operator. In the following example the ButtonClicked event is used to start matching against the purchase order database.

**Example. Sample Match in Validation**

```

Private Sub ValidationForm_ButtonClicked(ByVal ButtonName As String, ByVal pXDoc As
CASCADELib.CscXDocument)
    Dim sPONumber As String
    sPONumber = pXDoc.Fields.ItemByName("PONumber").Text
    Select Case ButtonName
    Case "Button0"
        Dim nMatchedRowCount As Long
        nMatchedRowCount = Match(pXDoc, "Base", "LIMLoc", "POLineItems", sPONumber, "")
    End Select
End Sub

```

**Using Match Function During Server Processing**

For documents that have table data available or for which the table was extracted by another locator you can use the Match function (for example, within the Document\_AfterExtract event) to match the existing table results against the purchase order database.

**Example. Sample Match in Server**

```

Private Sub Document_AfterExtract(ByVal pXDoc As CASCADELib.CscXDocument)
    ' get the PONumber and/or the VendorID from another field
    Dim sPONumber As String
    sPONumber = pXDoc.Fields.ItemByName("PONumber").Text
    Dim sVendorID As String
    sVendorID = pXDoc.Fields.ItemByName("VendorID").Text
    If Len(sPONumber) > 0 Or Len(sVendorID) > 0 Then
        Match pXDoc, "Base", "LIMLoc", "POLineItems", sPONumber, sVendorID
    End If
End Sub

```

# Script Samples

The following script samples are provided:

- [Set a Field's Confidence Threshold from a Script Variable](#)
- [Skip Layout Classification After 2nd Page](#)
- [Reclassify if the classification result has a specific value](#)
- [Find Most Recent Date](#)
- [Dynamically Suppress Orientation Detection for Full Page OCR](#)
- [Add an Alternative](#)
- [Classification by Blackness for Single Classes](#)
- [Classification by Graphical Lines](#)
- [Accepting Empty Fields](#)
- [Formatting Negative Amounts](#)
- [Access DPI](#)
- [Suppress OCR After Third Page](#)
- [Use Project Execution Mode](#)
- [Accessing Preprocessed Zones Images of Advanced Zone Locator](#)
- [Turn off Statistics](#)
- [Getting the root XFolder object by XDocument](#)

## Set a Field's Confidence Threshold from a Script Variable

This example shows how to use a script variable to set a field's confidence threshold.

Script variables are stored in an XML file external to the project, which can be updated separately. For this example, a script variable named "Threshold" has been configured. For more information, refer to the *Help for Project Builder*.

```
Option Explicit
' Project classification script
Private Sub Document_BeforeExtract(ByVal pXDoc As CASCADELib.CscXDocument)

    Dim sThreshold As String

    sThreshold = Project.ScriptVariables.ItemByName("Threshold").Value

    Project.ClassByName("ClassA").Fields.ItemByName("Surname").LocatorThreshold =
    CDb1(sThreshold)

End Sub
```

## Skip Layout Classification After Second Page

This example shows how to skip the layout classification after the second page. You can restrict the classification of a multi page document to the first page by selecting the "Classify only first page" option in the project settings. It may happen that such a document, for which only the first page is

processed, remains unclassified when, after the first page, the values for minimum confidence and minimum distance have not been reached. When you select “Classify each page,” the classification is performed for each page of a multi page document until a classification result is ascertained or all pages are processed. If you want to stop the layout classification after the second page of the document, select the “Classify each page” option in the project settings, insert the Document\_AfterClassifiy image into the edit area of the project classification script, and insert a condition to check the page number and to exit the loop over all pages.

```
Option Explicit
' Project classification script
Private Sub Document_AfterClassifyImage(pXDoc As CASCADELib.CscXDocument, ByVal PageNr As Long,
    bExit As Boolean)
    If PageNr >= 2 Then
        bExit = True
    End If
End Sub
```

## Reclassify if the classification result has a specific value

This example shows how to reclassify the current document to a different class. To avoid endless loops, the use of the reclassify method is limited to once per classification setup and once per extraction step.

```
Option Explicit
' Project classification script
Private Sub Document_AfterClassifyXDoc(pXDoc As CASCADELib.CscXDocument)
    If pXDoc.ExtractionClass = "ClassA" Then
        pXDoc.Reclassify("ClassB")
    End If
End Sub
```

## Find most recent date

To find the most recent date on a document, a format locator and a script locator can be combined. The format locator finds all possible dates on the document as alternatives. The script locator loops over all alternatives of the format locator (all possible dates) and picks the newest date and copies this as its own alternative. A date formatter is used to get the actual date value from the date string.

This might be useful to find the invoice date on a document, which is after the order and ship date, but before the due date. Of course it works only for current invoices, not for old documents.

Please take care to adjust the name of the format locator, the script locator and the date formatter before running the script.

```
Private Sub NewestDate_LocateAlternatives(pXDoc As CASCADELib.CscXDocument, pLocator As
    CscXDocField)
    Dim pDate As CscXDocField
    Dim pDateAlt As CscXDocFieldAlternative
    Dim i As Long
    Dim count As Long
    Dim MaxDate As Date
    Dim CurDate As Date
    Dim pTmpFeld As CscXDocField
    Dim MaxDateIndex As Long
    Dim pNewAlternative As CscXDocFieldAlternative

    MaxDate = Now - 3000
    MaxDateIndex = -1

    ' use actual Name of Format-Locator here, from where you want to take alternatives
    Set pDate = pXDoc.Locators.ItemByName("Date")
    count = pDate.Alternatives.Count
```

```

For i = 0 To count-1
    Set pDateAlt = pDate.Alternatives(i)
    Set pTmpFeld = New CscXDocField
    Set pTmpFeld = New CscXDocField
    pTmpFeld.Text = pDateAlt.Text
    StandardDatumsFormatierer.FormatField pTmpFeld

    If pTmpFeld.DateFormatted = True Then
        ' look for newest date before today
        If pTmpFeld.DateValue > MaxDate And pTmpFeld.DateValue <= Now Then
            ' remember this value
            MaxDateIndex = i
            MaxDate = pTmpFeld.DateValue
        End If
    End If
End If
Next i

' copy alternative with newest date before today
If MaxDateIndex > -1 Then
    Set pDateAlt = pDate.Alternatives(MaxDateIndex)
    pLocator.Alternatives.Create
    Set pNewAlternative = pLocator.Alternatives(0)

    ' copy all values
    pNewAlternative.Text = pDateAlt.Text
    pNewAlternative.Left = pDateAlt.Left
    pNewAlternative.Top = pDateAlt.Top
    pNewAlternative.Width = pDateAlt.Width
    pNewAlternative.Height = pDateAlt.Height
    pNewAlternative.PageIndex = pDateAlt.PageIndex
    pNewAlternative.Words.Append(pDateAlt.Words(0))
    pNewAlternative.Confidence = pDateAlt.Confidence
End If
End Sub

```

## Dynamically Suppress Orientation Detection for Full Page OCR

The Automatic Rotation option for full text OCR applies to the entire project. It is not possible to switch off orientation for single classes. The following script shows a workaround, which enables you to switch off orientation detection for OCR. The usage of the script requires that you can classify the document without OCR, e.g. layout classifier.

```

Private Sub Document_AfterClassifyXDoc(pXDoc As CASCADELib.CscXDocument)
    Dim i As Long
    If pXDoc.ExtractionClass = "MyClass" Then
        ' change orientation for all pages from unknown to NoRotation
        ' this prevents the OCR from doing orientation detection
        For i = 0 To pXDoc.CDoc.Pages.Count - 1
            pXDoc.CDoc.Pages(i).Rotation = Csc_RT_NoRotation
        Next i
    End If
End Sub

```

## Add an Alternative

To add an alternative to a script locator you have to set the text of the alternative and the confidence. The code snippet shows the script to add an alternative from a script locator named 'LocatorName'. The text of the alternative is "text of alternative" and the confidence is set to 20%.

```

Private Sub LocatorName_LocateAlternatives(ByVal pXDoc As CASCADELib.CscXDocument, ByVal
pLocator As CscXDocField)
    Dim oAlt As CscXDocFieldAlternative
    Set oAlt = pLocator.Alternatives.Create()
    oAlt.Text = "text of alternative"

```

```

    oAlt.Confidence = 0.2
End Sub

```

## Classification by Blackness for Single Classes

Some documents just cannot be classified by layout or content, because they don't contain a typical layout or content. This script sample shows how to classify a document by blackness. This might be useful for scanned photos, which normally appear very dark on the scanned image. The script calculates the average blackness on the image and checks also that the black regions are well distributed over the document. This should avoid conflicts with logos or graphical elements which might also generate dark regions but only on a single place of the image.

The function DetectBlackImage works on the first 3 pages of a document. Internally it calls DetectBlackImageOnPage, which works on a single page.

It can be called in the AfterClassifyXDoc event. It is assumed that 'Pictures' is a valid class name of the actual project.

```

Private Sub Document_AfterClassifyXDoc(pXDoc As CASCADELib.CscXDocument)
    If DetectGraphicLines(pXDoc) = True Then
        pXDoc.Reclassify("Pictures")
    Exit Sub
    End If
    '...
End Sub

Private Function DetectBlackImage(pXDoc As CASCADELib.CscXDocument) As Boolean
    Dim i As Long
    Dim count As Long
    Dim bResult As Boolean
    ' search for photos on the first 3 pages only
    count = pXDoc.CDoc.Pages.Count
    If count > 3 Then
        count = 3
    End If
    For i = 0 To count - 1
        bResult = DetectBlackImageOnPage(pXDoc.CDoc.Pages(i).GetImage())
        If bResult = True Then
            DetectBlackImage = True
            Exit Function
        End If
    Next i
    DetectBlackImage = False
End Function

Private Function DetectBlackImageOnPage(pImage As CscImage) As Boolean
    ' detects dark regions on a page
    ' this is used to detect class "Foto"

    Dim TileHeight As Long
    Dim XStart As Long
    Dim YStart As Long
    Dim x As Long
    Dim y As Long
    Dim dBlackness As Double
    Dim BlackTileCount As Long

    ' divide the image in 5 * 7 tiles (ignoring 1/2 tile as border)
    ' we have to check 4*6 tiles
    TileWidth = pImage.Width / 5

    TileHeight = pImage.Height / 7

    YStart = TileHeight / 2
    BlackTileCount = 0

```



```

For y = 0 To 5
  XStart = TileWidth / 2
  For x = 0 To 3
    dBlackness = pImage.GetBlackness(XStart, YStart, TileWidth, TileHeight)
    If dBlackness > 0.4 Then
      BlackTileCount = BlackTileCount + 1
    End If
    XStart = XStart + TileWidth
  Next x
  YStart = YStart + TileHeight
Next y

If BlackTileCount > 3 Then
  DetectBlackImageOnPage = True
Else
  DetectBlackImageOnPage = False
End If

End Function

```

## Classification by Graphical Lines

Some documents just cannot be classified by layout or content, because they don't contain a typical layout or content. This script sample shows how to classify a document by graphical lines. This might be useful for scanned charts or printed diagrams, containing a grid pattern as a background. The script calculates the number of vertical and horizontal lines on the image. Depending on some thresholds these numbers are used to make the classification decision.

The function DetectGraphicLines works on the first 3 pages of a document. Internally it calls DetectGraphicLinesOnPage, which works on a single page.

It can be called in the AfterClassifyXDoc event. It is assumed that 'Charts' is a valid class name of the actual project.

A reference to Kofax Cascade Forms Processing 2.0 must be added to the script sheet where that function is implemented.

```

Private Sub Document_AfterClassifyXDoc(pXDoc As CASCADELib.CscXDocument)
  If DetectGraphicLines(pXDoc) = True Then
    pXDoc.Reclassify("Charts")
  Exit Sub
End If
'...
End Sub

Private Function DetectGraphicLines(pXDoc As CASCADELib.CscXDocument) As Boolean
  Dim i As Long
  Dim count As Long
  Dim bResult As Boolean
  ' search for hor. and vertical lines on the first 3 pages only
  count = pXDoc.CDoc.Pages.Count
  If count > 3 Then
    count = 3
  End If
  For i = 0 To count - 1
    ' if we detect enough graphic lines on any of the first 3 pages, return TRUE
    bResult = DetectGraphicLinesOnPage(pXDoc.CDoc.Pages(i).GetImage())
    If bResult = True Then
      DetectGraphicLines = True
      Exit Function
    End If
  Next i
  DetectGraphicLines = False
End Function

```

```

Private Function DetectGraphicLinesOnPage(pImage As CscImage) As Boolean
    ' counts horizontal and vertical lines on a page
    ' this is used to detect class "Zeichnungen"
    Dim pLinesDetection As CscLinesDetection
    Dim xLeft As Long
    Dim xWidth As Long
    Dim yTop As Long
    Dim yHeight As Long

    ' check color format
    If pImage.BitsPerSample <> 1 Or pImage.SamplesPerPixel <> 1 Then
        DetectGraphicLinesOnPage = False
        Exit Function
    End If

    Set pLinesDetection = New CscLinesDetection
    ' setup parameters for lines detection
    pLinesDetection.DetectHorCombs = False
    pLinesDetection.DetectHorDotLines = False
    pLinesDetection.DetectHorLines = True
    pLinesDetection.DetectVerLines = True
    pLinesDetection.MinHorLineLenMM = 40
    pLinesDetection.MinVerLineLenMM = 40
    ' start lines detection, skip a border of 5%
    xLeft = pImage.Width * 0.05
    xWidth = pImage.Width * 0.9
    yTop = pImage.Height * 0.05
    yHeight = pImage.Height * 0.9
    pLinesDetection.DetectLines pImage, xLeft, yTop, xWidth, yHeight

    ' we require more than 8 hor. and vertical lines to return TRUE
    If (pLinesDetection.HorLineCount > 8 And pLinesDetection.VerLineCount > 8) Then
        DetectGraphicLinesOnPage = True
    Else
        DetectGraphicLinesOnPage = False
    End If
End Function

```

## Accepting An Empty Field

During processing there are two points in time when you can accept an empty field and change the field status appropriately:

- in Server by setting the field's property `ExtractionConfident`
- in Validation by using a script validation rule

### Example. AfterExtract Event Sample Code

The following code sample shows how to set the field status for “TestField” to valid if the field is empty.

```

Private Sub TestField_AfterExtract(ByVal pXDoc As CASCADELib.CscXDocument, ByVal pField As
    CASCADELib.CscXDocField)

    If len(pField.Text)=0 Then
        pField.ExtractionConfident=True
    End IF
End Sub

```

### Example. Sript Validation Rule Sample Code

The following single field script validation rule sets the field status to “valid” if the field is empty.

```

Private Sub ScriptValRuleEmptyField_Validate(pValItem As CASCADELib.ICscXDocValidationItem,
    ErrDescription As String, ValidField As Boolean)

Dim Field As Object
    If Len(pValItem.Text) = 0 Then
        ValidField = True
    Else
        ValidField = False
    End If
End Sub

```

## Formatting Negative Amounts

The standard amount formatter does not support negative amounts. To preserve the minus sign with negative amounts a script formatter with the following script code can be used.

- 1 Add a new script formatter.
- 2 Change the type of the formatter to amounts.
- 3 Insert the following script code into the event method of the formatter.

Take care to use the correct function name, according to the name of your script formatter:

```

Private Sub AmountScript_FormatdoubleField(ByVal FieldText As String, FormattedText As
    String, ErrDescription As String, _ ValidFormat As Boolean, ByRef DoubleVal As Double, ByRef
    DoubleFormatted As Boolean)

Dim pTmpField As New CscXDocField

pTmpField.Text = FieldText

' use the standard amount formatter first
' replace "StandardBetragsFormatierer" with the name of you normal amount formatter
ValidFormat = StandardBetragsFormatierer.FormatField(pTmpField)
DoubleVal = pTmpField.DoubleValue
DoubleFormatted = pTmpField.DoubleFormatted
FormattedText = pTmpField.Text

If ValidFormat = False Then
    ErrDescription = pTmpField.ErrorDescription
End If

' check for minus sign inside original text
' you could also be more strict and allow only the first or the last character to be a minus
sign
' but this might cause problems if the currency symbol is also there
If InStr(FieldText, "-") > 0 Then
    ' make it negative
    FormattedText = "-" +FormattedText
    DoubleVal = - DoubleVal
End If

End Sub

```

## Access DPI

If you want to access the DPI (resolution: dots per inch, normally 300 for scan and 200 for fax) you can use the following code:

```

Dim XRes As Integer, YRes As Integer

XRes = pXDoc.CDoc.Pages(PAGENUMBER).XRes ' for horizontal resolution

```

```
YRes = pXDoc.CDoc.Pages(PAGENUMBER).YRes ' for vertical resolution
```

## Suppress OCR After Third Page

This example shows how to skip OCR for all pages after the third page. All documents have three or less pages have a complete OCR, documents having more than three pages have only OCR on the first three pages.

```
Private Sub Document_BeforeProcessXDoc(pXDoc As CASCADELib.CscXDocument)
    Dim i As Long
    Dim Count As Long
    Count = pXDoc.CDoc.Pages.Count

    ' suppress OCR for all pages after 3
    For i = 3 To Count - 1
        pXDoc.CDoc.Pages(i).SuppressOCR = True
    Next i

End Sub
```

## Use Project Script Execution Mode

The property `Project.ScriptExecutionMode` contains the information which module is running the script. For example, this information can be used for differentiating the script dependent on the executing module.

```
...
Select Case Project.ScriptExecutionMode
    Case CscScriptModeServer
        ' this part is only executed during server processing
    Case CscScriptModeServerDesign
        ' this part is executed during server processing in Project Builder
    Case CscScriptModeUnkown
        ' this part is executed if the script execution mode is unknown
    Case CscScriptModeValidation
        ' this part is only executed during validation
    Case CscScriptModeValidationDesign
        ' this part is only executed during validation design in Project Builder
End Select
...
```

## Accessing Preprocessed Zones Images of Advanced Zone Locator

By default these zone images are not available from script to optimize memory consumption during document processing. If you activate the property “SaveXDocImages” of the Advanced Zone Locator in the “BeforeExtract” event you can access the registered coordinates, the original coordinates, and the preprocessed image for each defined zone. This information is only accessible after extraction and is not saved to disk, you can access this information as shown in the “AfterExtract” event. You have to add an additional reference for the Advanced Zone Locator (Kofax Cascade Advanced Zone Locator 2.0) to the script sheet where you insert the following script code.

```
Private Sub Document_BeforeExtract(pXDoc As CASCADELib.CscXDocument)

    Dim oClass As CscClass
    Dim oLocDef As CscLocatorDef
    Dim oAdvZoneLoc As CscAdvZoneLocator

    Set oClass = Project.ClassByName("MyClassName")
    Set oLocDef = oClass.Locators.ItemByName("MyAdvancedZoneLocator")
    Set oAdvZoneLoc = oLocDef.LocatorMethod
```

```

        oAdvZoneLoc.SaveXDocImages = True

End Sub

Private Sub Document_AfterExtract(pXDoc As CASCADELib.CscXDocument)

    Dim oRep As CscXDocRepresentation
    Dim oXDocImg As CscXDocImage
    Dim oImg As CscImage

    Set oRep = pXDoc.Representations.ItemByName("AdvZoneLoc") ' this is the image of the 0.
defined (first) zone in the advanced zone locator
    Set oXDocImg = oRep.Images(0)
    Set oImg = oXDocImg.Image

    oImg.Save("C:\xyz.tif")

End Sub

```

## Turn off Statistics

By default all fields of a document are written to the statistics database. If you have fields that are of no interest for the statistics you can exclude them. In the “Document\_Validated” event (this event is also raised for documents that are classified or reclassified during validation) you can set the property “UseForStatistics” to FALSE for those document fields you do not want to save to the statistics database.

```

Private Sub Document_Validated(pXDoc As CASCADELib.CscXDocument)
    Dim n As Long
    Dim oXField As CscXDocField
    For n = 0 To pXDoc.Fields.Count - 1
        Set oXField = pXDoc.Fields(n)
        Select Case oXField.Name
            Case "FieldOfNoInterest1"
                oXField.UseForStatistics = False
            Case "FieldOfNoInterest2"
                oXField.UseForStatistics = False
            Case "FieldOfNoInterest3"
                oXField.UseForStatistics = False
        End Select
    Next n
End Sub

```

## Getting the Root XFolder Object by Given XDocument

Define the function GetRootFolder in the project script so that it is accessible from every other script sheet. This function searches recursively for the the XRootFolder object and if found it returns it.

```

Public Function GetRootFolder(pXFolder As CASCADELib.CscXFolder) As CASCADELib.CscXFolder
    If pXFolder.IsRootFolder Then
        Set GetRootFolder = pXFolder
    Else
        Set GetRootFolder = GetRootFolder(pXFolder.ParentFolder)
    End If
End Function

' Call the function GetRootFolder in events having only the XDocument as parameter
...
Dim pXRootFolder As CascadeLib.CscXFolder
Set pXRootFolder = GetRootFolder (pXDoc.ParentFolder)
...

```

---

# Using Help

The Help for this module contains two panels. The left panel contains three tabs:

- [Contents](#)
- [Index](#)
- [Search](#)

The main frame contains the individual [Help topics](#).

## Navigating the Help

There are several ways to navigate between topics and books:

- Use the Contents tab to expand books and select a topic to view.
- Use the breadcrumb links in the header of each topic to move to a parent topic.
- Use links within a topic to go to related content elsewhere in the Help.



## Contents

The Contents contains several expandable books that contain one or more books or topics.

Each book contains related content. For example, the top-level Overview book contains several topics which provide an introduction to Kofax Transformation Modules. Other books include topics and several books on how to use this module.

When you click on a book, it displays a general overview of what that book contains. In most cases, the top level topic of a book will also contain links to important topics and information contained within the book.

### ► To view the contents of a book or topic

- 1 Click on the Contents tab.
- 2 Expand  or collapse  the books as necessary.
- 3 Click on the desired topic or book to view its contents.

## Glossary

A Glossary of important terms, arranged in alphabetic order can be found at the bottom of the Contents tab. The terms listed here provide information that is not available elsewhere in the Help.

### ► To view the glossary

- 1 Click on the Contents tab.
- 2 Click on the Glossary topic.
- 3 Scroll through the glossary until you find the desired term.

## Index

The index contains a list of important terms and phrases arranged in alphabetic order. Each term is linked to a topic one or more topics where more information can be found.

You can also search the contents of the index by typing a keyword.

### ► To search for an index keyword

- 1 Type a keyword in the Type in the keyword to find box.
- 2 Press ENTER.
- 3 Double-click on the desired term to view the topic contents.

### ► To open an index term topic

- 1 Scroll through the index to find the desired term.
- 2 Double-click on the desired term to view the topic contents.

## Search

The search tab enables you to search the help content for specific keywords. Any topics that contain the specified keyword can be opened.

### ► To search for a keyword

- 1 Type the keyword in the Type in the keyword to find box.
- 2 Click List Topics.
- 3 Double-click on the desired topic to view its contents.

## Help Topics

In order to maximize space and minimize reading time, several elements on individual topics have been configured to only show when a user wishes to see that content.

## Procedures

Procedures, which contain multiple steps to perform an action are collapsed by default. A collapsible procedure can be expanded by clicking on the Arrow ► icon. Clicking it a second time collapses the procedure.

### ► An example of a Procedure

**Figure 10.1. Collapsed Procedure with Multiple Steps**

### ▼ An example of a Procedure


1. Step one.
2. Step two.
  - a. Sub step a.
  - b. Sub step b.
3. Step three.


**Figure 10.2. Expanded Procedure with Multiple Steps**


Procedures with a single step are expanded by default and cannot be collapsed. These look similar to multi-step procedures, but are not preceded with an icon.

**A procedure with one step**, appears on a single line, is open by default, and cannot be collapsed.


**Figure 10.3. Single-Step Procedure****Examples and Other Collapsed Text**

In addition to procedures, lengthy examples and supplementary text can be collapsed. Instead of using the arrow symbol, these sections display a title next to an Expand  button.

To expand one of these sections, click on the expand button. Expanded sections can be collapsed by clicking on the Collapse  button.

 **For example...**

**Figure 10.4. Collapsed Example**

 **For example...**

Collapsible text can contain an example, a list of prerequisites, or instructions on what to do next. Each of these can contain supplementary information, graphics, code snippets, as well as other procedures.

**Figure 10.5. Expanded Example****Notes, Tips, Important, and Caution**

Useful information through the Help is displayed in “Notes”, suggestions are explained in “Tips”, important information is displayed as “Important”, and common mistakes are showed as “Caution”.



## **Related Documentation**

In addition to this Kofax Transformation Modules Script reference document, there is also available the Object Reference Help available from the Project Builder interface.

---

# Technical Assistance for Your Kofax Product

Support for your Kofax product is provided by your primary application support provider, which is specified as part of the maintenance agreement associated with your purchase. Please contact your Kofax application support provider for technical assistance with your Kofax product.

For more information about your product, visit the [Kofax Support pages](#) for:

- Product information and release news
- Access to the Kofax Knowledgebase
- Access to the online Web Incident Management Systems for eligible customers
- Downloadable product documentation

Before contacting your Kofax application support provider, please gather the following information where applicable:

- Product name, version, and serial number
- Log files
- Product license
- Exact error message(s)
- Reproduction scenario

---

# Glossary

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [K](#) [L](#) [M](#) [O](#) [P](#) [R](#) [S](#) [T](#) [U](#) [V](#)

## A

Activation	The process of enabling your Kofax Transformation Modules licensing.
Adaptive Feature Classifier	The Adaptive Feature Classifier (AFC) performs an analysis of the textual representation of a document and automatically classifies it into a class.
Address Evaluator	An evaluator that compares address fields that are extracted with a zone locator (for example zip code, city or street) to the values in a database, and if required corrects the field data.
Alternative	See <i>Detected Alternatives</i>
Amount Formatter	A group of related settings that govern how amounts are formatted after extraction. For example, ensuring that a total value adds up to its component values.
Application Programming Interface	A set of routines used to direct the performance of procedures in an application group. Allows custom module developers to take advantage of the many capabilities build into Kofax Transformation Modules.
Autoclustering	Automatically clusters images into similar groups to speed up processing. See <i>Image clustering</i> .
Automatic Foldering	After foldering is set up, folder levels and the script programming and if needed, folder fields and validation rules for folders, you can apply foldering to a set of test documents. Folders are created and the documents resorted into the corresponding folder. If defined the folder fields are extracted and validated.

## B

Bar code	A special identification code printed as a series of bars, dots, or other shapes. A bar code has build in error checking, and may contain text, number, or both.
Batch	A Batch is a group of documents (or folders) that are run through Kofax Capture based on a Batch Class.
Batch Class	<p>A Batch Class describes how the documents in a batch are processed by Kofax Capture. It includes which Kofax Transformation Modules and Kofax Capture modules are used to process images, and in what order. It also determines how images are separated into documents, how forms are identified and how images are cleaned up.</p> <p>Each batch class can contain one or more classes. Each class can contain one or more form types. This ensures you are able to have different kinds of documents or forms within a single batch.</p>
Best Fit	This Document Viewer option fits the selected image to the page.
Blank Out Regions	This option enables you to remove non-relevant text areas or characters from a document so they are ignored by a locator.

## C

Class	A class is a category that can be assigned to a document during classification.
Classification	<p>The process of determining the class to which a document belongs.</p> <p>There are two basic groups of classifiers. The first group uses the layout of a document to determine its class. These are layout classifiers.</p> <p>The second group uses extracted text to determine the class of a document. These are called content classifiers.</p> <p>The Layout Classifier is an layout classifier, while the Adaptive Feature Classifier (AFC) and the Instruction Classifier are content classifiers.</p>
Classifier	A classifier is an algorithm that classifies a document based on certain characteristics of that document, such as its physical layout.
Clustering	See <i>Image clustering</i> .
Confidence	A number returned by the recognition engine that indicates the percentage of certainty for the reported results.
Context Menu	A context menu is displayed when you right-click your mouse.
Content Classification	Makes use of the extracted content of a document in order to classify it.
Correction	<p>Correction is a module that allows keyboard operators to take data that was extracted in Server and correct any information that was rejected or fails validation rules.</p> <p>An optimized interface allows fast correction of single characters using a single keystroke.</p>

## D

Database	A database can be relational or fuzzy. A fuzzy database is a .csv file that contains delimited data that can be used by locators during extraction. A relational database can be a Microsoft SQL Server, Oracle Server, or ODBC connection.
Date Formatter	A group of related settings that govern how dates are formatted after extraction. For example, processed documents may contain dates in different formats. In order to store the extracted date in a back-office system, they must all be consistent. The Date Formatter converts any date to a specific format so they are consistent.
Default Validation Form	The default validation form shows all extraction fields defined for a class on a form using the standard layout 1. This layout shows all fields and tables defined for the class. For a simple field a label and a mini viewer are shown left of the field. The default validation form is used in Test Validation or the Validation module when no validation form is designed for a class; or if no form can be inherited from a parent class.
Detected Alternatives	A locator usually does not return only one relevant item that can be assigned to a field. Instead, it creates sets of alternatives that are weighed according to their confidence level. During the evaluation step, the locator compares the alternatives based on additional settings you provide in the locator properties. Based on that evaluation, each alternative is assigned a confidence value between 0% and 100%. Only one alternative is returned as the result.
Dictionary	A dictionary is a text file that contains values that can be used to aid in extraction. A dictionary can only have one entry per line and can contain additional replacement values separated by a delimiter.

	For example, a dictionary can contain a list of calendar months. Each month can also be paired with a numerical value. This means that when extraction runs, the alphabetic name of the month is replaced with the numerical value.
Document Review	Document Review is a module that allows keyboard operators to ensure that every document in a batch contains the correct pages and has the correct classification.  Users are presented with a list of problems that have failed Document Review rules and must be fixed.
Document Separation	The process of taking on large document and splitting it into several smaller documents using any of the Kofax Transformation Modules classification methods.
Dongle	See <i>Hardware key</i> .
Dynamic Zone Adjustment	An option in the Advanced Zone Locator that automatically increases the size of a zone so it includes all necessary content. This means that if a zone is created on one image so all content is well within the boundaries, the zone can be automatically adjusted if another image's content is slightly shifted.  Dynamic zone adjustment can done done in four directions. To the top, bottom, left and right of a zone.
<b>E</b>	
Evaluator	A class of engines that compares multiple possible results to determine the best choice among them.
Extraction	The process in which a recognition engine identifies information on a document that can later be used to identify specific characters.
<b>F</b>	
Field	A field is a piece of extracted data.
Field Formatter	A group of related settings that govern how items (such as dates or monetary amounts) are formatted after extraction.
Folder	A folder is a directory within a batch that is used to sort the batches documents for additional criteria, for example a supplier name or a customer ID.  For a Kofax Transformation Modules project you have to enable foldering and define folder levels. For each folder level you can define a sub folder, folder fields and validation rules.  The folder and its sub folders, in a batch is always an instance of a folder level that is defined in the project.
Foldering	If foldering is defined, you can sort documents of a batch for similar criteria, for example a customer ID, supplier name. So called folder are created and the documents are sorted to those folders. Similar to “normal” extraction fields, defined for a class, you can define folder fields. Those fields are extracted after the resorting and if available, validation rules are applied.
Folder Level	To be able to sort the documents according to similar criteria like for example, the customer ID, you define folder level for a project. For each folder level you can only define one sub folder level. For each folder level you can create folder fields and validation rules.  If foldering is enabled, the server creates the folders for a batch, resorts the documents and extracts the folder fields. A batch may now contain several

folders, but at the same hierarchy level you always have the same type of folder, for example, one folder for the documents of supplier A, another folder for supplier B and a third folder for supplier C.

Formatting

Formatting

## G

Generic Learning

A learn-by-example technology that uses a knowledge base to extract information from invoice documents. Extraction is based on keywords, data format and position. These are generalized so they can be applied to unseen images from new suppliers.

Generic Online Learning

Performed on-the-fly without the need for administrator intervention. Keyboard operators identify documents where extraction was not successful and they are automatically processed by the Knowledge Base Learning Server.

## H

Hardware Key

A device that plugs into a computer to prevent the unlicensed use of Kofax Transformation Modules.

## I

Image Clustering

A process in which documents are grouped by geometric similarity in order to speed up layout classification setup.

Installation

The act of copying essential application files to your system and preparing it for use. The installation process is usually automated by an installation or setup application or by a wizard.

Instruction Classification

A process in which documents are grouped by geometric similarity in order to speed up layout classification setup.

Instruction Classifier

The instruction classifier searches for given phrases in the text representation and uses predefined rules to classify it.

## K

Keying Modules

Designed for operator use to ensure that all documents have been successfully classified and extracted by the automatic server modules.

Knowledge Base

A repository of information about invoice documents that contains data on keywords, data formats, positions, and supplier information (for specific knowledge bases).

## L

Language Pack

A file used to change the application language for Kofax Transformation Modules.

Layout Classifier

An layout classifier that performs image-based classification by analyzing the graphical elements of an image without the need for OCR.

Locator

A configurable option that enables you to extract different types of data from a document.

## M

Menu

A list of options that allow a user to make a selection in order to perform a desired action.

Multiple Validation Steps	In a workflow, validation capabilities may be needed several times, for example that different users can validate subsets of fields only. Therefore the Validation module can be added up to five times to the queue of the Kofax Capture batch class.
<b>O</b>	
ODBC	A standard method for connecting to different types of databases, independent of programming languages or operating system used.
Online Learning	See <i>Specific online learning</i> or <i>Generic online learning</i> .
Optical Character Recognition	The ability of software to recognize printed characters and translate them into computer-readable data.
Optical Mark Recognition	The ability of software to recognize marked areas on forms and convert the date to a format that can be processed by the software. The marks are typically created by filling in a circle or marking a check in a check box on a preprinted form.
<b>P</b>	
Percentage Formatter	A group of related settings that govern how percentages are formatted after extraction.
Problem (Document Review)	One or more failed Document Review rules.
<b>R</b>	
Recognition	The process by which computers can interpret handwritten characters or other marks in an image and convert them to computer-readable data.
Regular Expression	A method of describing data in an abstract way to recognize patterns within textual data.
<b>S</b>	
Server	Automatically separates a batch into documents and then classifies and extracts information from each document. A second instance of server can be used to automatically extract information from already separated and classified documents.
Separation	The process by which individual pages are arranged into documents.
Solution Integrator	The person responsible for fine-tuning Kofax Transformation Modules to the specific documents being processed. The options configured by the solution integrator determine how documents are classified and extracted, and also affect the options available in the user interactive modules.
Specific Learning	A learn-by-example technology which uses a knowledge base to extract information from invoice documents from known suppliers. Once a few samples from a supplier have been trained, all invoices from that supplier can then be extracted with high confidence.
Specific Online Learning	Specific learning performed on-the-fly without the need for administrator intervention. Keyboard operators identify documents where extraction was not successful, and they are automatically processed by the Knowledge Base Learning Server, which maintains a dynamic specific knowledge base.

## T

### Training Set

A group of documents used to configure a project to improve classification and extraction results. From a training set you can create a knowledge base to be reused for other projects.

## U

### Unconfident

A class, field, or character with a confidence value below the threshold for it to be considered valid. Unconfident classes, fields, and characters require user attention.

## V

### Validation

Reviews extraction results for an entire document and ensures that all information is valid. Documents can also be marked for online learning. Additional validation steps can be used to review a subset of extraction results in configurations where validation is performed in multiple steps.

### Verification

Verification allows comprehensive verification of critical data fields using a keyboard-optimized interface. Verification is processed on documents that have passed the final validation step. Following modes can be configured for the verification: confirmation (by pressing a single key to accept the data), blind double keying (a more comprehensive verification technique) and reviewing the fields in a read-only state.