opentext™

OpenText™ Intelligent Capture

Scripting Guide

This guide is intended for developers and describes how to change the default behavior of Intelligent Capture with custom code, including the various programming and scripting types as well as the various APIs available for each scripting type.

ECPCORE210400-PSC-EN-01

OpenText™ Intelligent Capture
Scripting Guide
ECPCORE210400-PSC-EN-01
Rev.: 2021-Sept-13

This documentation has been created for OpenText™ Intelligent Capture CE 21.4.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Chapter 1

Overview

Users can change the default behavior of Intelligent Capture with custom code. The following sections describe the various programming and scripting types, and list the various APIs available for each scripting type.

1.1 Programming and Scripting Types

Intelligent Capture includes the following scripting types.

Profile Scripting
Use for document types, page-level image enhancements, task events, and batch creation. This type of script is meant to be reused and be independent of the process. Profile scripts cannot access the task scripting *API*s or events.

Task Scripting
Use for task and batch node manipulation. This type of script has access to IA Values and the batch. If it gets a document data block, then a task script can use profile script APIs to manipulate the object. It cannot use the profile scripting events or UI-related APIs.

Recognition Scripting
Use for customizing Classification and Identification to manipulate classification and pre-indexing results using VBA.

Client-Side Scripting
: Use for creating client scripts to automate tasks in capture processes. A client-side script is a DLL that runs as part of a module step within a CaptureFlow. Several modules support client-side scripting. To use client scripts, you create script actions and then associate them with specific events that are defined in each module. The occurrence of the event triggers execution of the script action.

Notes

- While Windows Presentation Framework (WPF) is recommended for all custom UI, Windows Forms can also be used, but only during button click events. The use of Windows Forms during other UI events can cause unwanted side effects in behavior.
- Client-side scripts, profile scripts, and Recognition scripts are handled directly by the modules that support them and do not require an extra step in the CaptureFlow.

Intelligent Capture also includes the following programming types:

Intelligent Capture REST Services
Intelligent Capture Real Time Services is a product offering based on Intelligent Capture REST Services, which are a set of RESTful web service interfaces that

custom client applications can use to call the services of the Intelligent Capture Server or the Module Server. An example of an Intelligent Capture REST Services client is Intelligent Capture Web Client.

1.2 List of Scripting Namespaces

The following table lists the Scripting namespaces available in Intelligent Capture and a brief description of each namespace.

Namespace	Description
`Emc.InputAccel.CaptureClient`	Provides access to client modules and the batch data it processes. This namespace is used by the .NET Code module.
`Emc.InputAccel.CaptureFlow`	Provides access to batch data from the CaptureFlow Script Editor in CaptureFlow Designer.
`Emc.InputAccel.ImageFilter`	Provides classes used to create custom image processing filters.
`Emc.InputAccel.UimScript`	Provides classes used by Document Type, image processing super filter scripting, task event handling, and batch creation scripting.
`Emc.InputAccel.DocumentumExport`	Provides classes that determine the set of Documentum objects and attributes that are standard for definition setup.
`Emc.InputAccel.DocumentumExport.Scripting`	Provides the Client-Side scripting interface to the Documentum Advanced Export module.
`Emc.InputAccel.DocumentumExport.Setup`	Provides classes that handle objects for Documentum Advanced Export in setup mode.
`Emc.InputAccel.NuanceOCR.Scripting`	Provides the Client-Side scripting interface to the NuanceOCR module.
`Emc.InputAccel.QuickModule.ClientScriptingInterface`	Provides the common Intelligent Capture Client-Side scripting interface.
`Emc.InputAccel.ScanPlus.Scripting`	Provides the Client-Side scripting interface to the ScanPlus and RescanPlus modules.
`Emc.InputAccel.WebServicesScripting`	Provides the Client-Side scripting interface to Web Services Input and Web Services Output client modules.

1.3 Choosing Between Document Type Scripting and Recognition Scripting

This section gives context on how to choose between Profile scripting and Recognition scripting.

In a process that uses Advanced Recognition client modules and modules added in 7.0 such as Completion and Extraction, designers will need to maintain two sets of scripts, one for the Recognition Scripting VBA and the other for the Profile Scripting .NET code. The design goal in these cases is to minimize or avoid the overlap between these two sets of scripts. The behavior of both types of scripting are described in the following cases.

1.3.1 New Recognition Projects used only for Classification and Extraction Modules

Scripts that are used for Classification (for example, a script that conditionally assigns a template to a document) are beyond the scope of a particular Document class; that is, they do not exist as part of Document Type scripting. Certain Extraction events, such as `Document_BeforeRecognition` are also beyond the Document Type scope. These events require VBA scripts for the Recognition project (DPP), and classification and recognition events, as was done in previous versions of Dispatcher. These scripts should not contain event handlers for the scope of a Document Type.

Event handlers for the scope of a Document Type must be created as Document Type scripts.

At runtime, the Classification and Extraction modules will each fire the same events as they did in previous versions of Dispatcher. After these events are complete, the Extraction module will call the appropriate API to validate the document, at which time the Document Type event handlers will run for the entire document.

Note: In the Completion module, only the Document Type events described in this document will run.

1.3.2 Migrated Projects (Classification and Extraction Only)

Previous Dispatcher projects can be imported to Recognition Designer and converted to Recognition projects. Imported projects that only use Classification and Extraction are similar to new Recognition projects. For imported projects, VBA code already exists for all event handlers, and the Document Type-related event handlers must be rewritten as Document Type scripts, and then those VBA event handlers must be removed from the imported Recognition project.

At run time, the Classification and Extraction modules will each fire the same events as they did in previous versions of Dispatcher. After these events are complete, the Extraction module will call the appropriate API to validate the document, at which time the Document Type event handlers will run for the entire document.

1.3.3 Projects Using Identification

In a process that uses Identification, if the Recognition Project is configured to use some of the fields as pre-indexing fields, the pre-indexed fields will have UI and validations that must run beyond the scope of the Document Type scripting. In this case, script duplication cannot be avoided as Identification must validate the fields using its VBA scripts and 7.0 client modules will revalidate the fields using its Document Type scripts. It is the script author's responsibility to ensure that these events behave consistently with one another.

The handling of scripts other than those for the pre-indexed fields is the same both at design and runtime, independent of whether the project is new or has been migrated from an existing Dispatcher project.

At runtime, Identification will fire the same events as in previous versions of Dispatcher; only those that have scope beyond a Document Type and those for the pre-indexed fields should have handlers and do work. After Identification is complete, the module will not fire the `FireValidateDocument` event as it is only intended for unattended modules.

1.3.4 Mapping of Recognition (Dispatcher) Scripting Events to Document Type Scripting Events

The table below shows the relationship between the Recognition Scripting event model and Document Type Scripting events.

Table 1-1: Recognition (Dispatcher) Scripting Events Mapped to Document Type Scripting Events

Recognition (Dispatcher) Scripting Events	Replacement with Document Type Scripting Events
Project Events	
`Project_Initialize`	`ScriptMain.ScriptLoaded`
`Project_Finalize`	`ScriptMain.ScriptUnloaded`
`Project_BeginTask`	`Script<`*`taskName`*`>.TaskLoad`
`Project_EndTask`	`Script<`*`taskName`*`>.BeforeTaskFinish`
`Project_BeforeTaskAbort`	`Script<`*`taskName`*`>.BeforeTaskFinish`
Indexing Family Events	
`IndexingFamily_Initialize`	In the `Script<Class>.DocumentLoad` event, use a static member to indicate if the load is the first for the Document Type.
`IndexingFamily_Finalize`	In the `Script<Class>.DocumentUnload`, use a static member to indicate if the load is the last for the Document Type.

Recognition (Dispatcher) Scripting Events	Replacement with Document Type Scripting Events
`IndexingFamily_BeforeDocumentRecognition`	None before OCR. `Script<Class>.DocumentLoad` executes immediately after OCR.
`IndexingFamily_AfterDocumentRecognition`	Scripted validation rules execute after OCR and field-level checks.
`IndexingFamily_EnterDocument`	`Script<Class>.FormLoad`
`IndexingFamily_EnterDocument`	None when switching docs in a task, then `Script<Class>.DocumentUnload` when the task ends.
`IndexingFamily_BeforeDocumentReject`	Use `Script<Class>.DocumentClosing`, although it does not enable the close to be cancelled. To enable the close to be cancelled use `Script<TaskName>.BeforeTaskFinish` instead.
`IndexingFamily_DocumentValidated`	None explicit. After each scripted validation rule, check all fields to see if they are all valid.
`IndexingFamily_PressCustomHotKey`	Create a button in the form, then assign a shortcut key to it. The shortcut will then fire the `Script<Class>.ButtonClick<Button>` event.
`IndexingFamily_ControlDocument`	None. Document status and the rules that must rerun are managed automatically by the script engine.
Field Events	
`<Field>_Enter`	`Script<Class>.EnterControl<Field>`
`<Field>_ValidateValue`	`Script<Class>.ExitControl<Field>`
`<Field>_BeforeRecognition`	None before OCR. The `Script<Class>.DocumentLoad` executes immediately after OCR.
`<Field>_AfterRecognition`	Scripted validation rules execute after OCR and field-level checks.
Table Events	
`<Table>_BeforeAddRow`	None. New rows can be deleted after it has been added.
`<Table>_AfterAddRow`	`Script<Class>.InsertRow`
`<Table>_BeforeDeleteRow`	`Script<Class>.DeleteRow`
`<Table>_AfterDeleteRow`	None. The row has already been deleted.

Recognition (Dispatcher) Scripting Events	Replacement with Document Type Scripting Events
Dispatcher Function Library	
`Math` functions	.NET`System.Math` namespace
`String` functions	.NET `System.String`, `System.Text.Regex`, and `<data_type>.TryParse`
`Date` functions	.NET `System.DateTime`
`GUI` functions	.NET `System.Windows.MessageBox`, custom pick list code
`TIFF` functions	None. Scripting cannot access these images.
`System` functions	.NET support for calling other DLLs, For example: `Environment.UserName` and `System.Net.Dns.GetHostName()`
`File` functions	.NET `System.IO` namespace
`Checkbox` functions	Read each checkbox field and calculate using .NET code

Chapter 2

Profile Scripting

2.1 Overview

You use profile scripting as follows:

- To extend the population and validation of document data.
- To provide additional control over how a set of image processing filters are applied.
- To handle task events.
- To provide control over batch creation.

2.2 Architecture

2.2.1 Class Structure

You create a single main class, named ScriptMain, that extends `Emc.InputAccel.UimScript.UimScriptMainCore` and any number of custom script classes that correspond to the scripting that you want to implement. Each custom script class contains the appropriate script code and the ScriptMain class loads the custom script classes. Each custom class script has the following characteristics:

Table 2-1: Custom Script Class Characteristics

Object Type	Class Name	Intelligent Capture Designer Specification	Namespace for Custom Class	Base Class
Document Type	Script<documentTypeName>	Document Type name	Custom.InputAccel.UimScript	Emc.InputAccel.UimScript.UimScriptDocument
Super Filter	Script<profileName>	Image Processor profile name	Custom.InputAccel.UimFilterScript	Emc.InputAccel.UimScript.UimScriptFilter
Task Event Type	Script<myClass>	In the applicable step's setup mode, *<myClass>* in the **Script Name** property.	Custom.InputAccel.UimTaskScript	Emc.InputAccel.UimScript.UimScriptTask

Object Type	Class Name	Intelligent Capture Designer Specification	Namespace for Custom Class	Base Class
Batch creation	ScriptMail<profileName>	**Standard Import Email Import** profile name	Custom. InputAccel. UimBatchScript	Emc.InputAccel. UimScript. UimScriptBatch Create
	ScriptFile<profileName>	**Standard Import File System** profile name		

2.2.2 DLL Structure

At runtime, the Intelligent Capture framework loads the scripts from DLLs. The profile script classes can be built into DLLs as follows.

- ScriptMain class

  This class must be built into `Custom.InputAccel.UimScript.dll`.

- Profile scripting classes

  These classes can be built into `Custom.InputAccel.UimScript.dll`, separate DLLs, or both. You could use the following naming convention: `Custom.InputAccel.UimScript.`*`<myName>`*`.dll` where *<myName>* is any descriptive name. For example, a DLL for tax form scripts could be named `Custom.InputAccel.UimScript.TaxForms.dll`.

  Note: Each custom script class name must be unique among all DLLs.

- `ISV.InputAccel.UimScript.dll` (for an assembly created by OpenText Global Technical Services or any other third-party developer)

2.3 Procedure

This section describes how to create, deploy, and debug profile scripts.

2.3.1 Creating Profile Scripts

This section details the specific actions the user must take to create a profile script. Script authors may make a copy of the sample Document Type and script solution and customize it to create their own profile script. Alternately, they can follow these steps to create and deploy profile scripts.

By default, profile script samples are located at: *`<path to client install>`*`\Client\src\Sample Capture System\ScriptSource\Profile Scripts`

To create profile scripts:

1. Create a Visual Studio project for a C# or Visual Basic Class Library

2. Edit project properties:

   a. Set the **default namespace** to `Custom.InputAccel.UimScript` if you are creating profile scripts in C#. If you are creating profile scripts in VB.NET, leave this field blank, and set the namespace for each class separately.

   b. Set the **Assembly name** to `Custom.InputAccel.UimScript` or `ISV.InputAccel.UimScript` as appropriate.

   c. Set **TargetFramework** to **4.8**.

   d. Ensure **Output Type** is **Class Library**.

   e. Set **Build output path** to *<Capture System>*`\bin`

3. Add a reference to *<path to client install>*`\Client\binnt\Emc.InputAccel.CaptureClient.dll`. This DLL resolves references to all context interface objects and classes needed by the profile script.

4. Implement a `ScriptMain` class for script assemblies. This class extends the `Emc.InputAccel.UimScript.UimScriptMainCore` class. `ScriptMain` must provide a constructor that takes no parameters. Optionally, it can override the following methods:

   - `ScriptLoad()`: Is called once during process execution lifetime and can include the global initialization code for all documents.
   - `ScriptUnload()`: Is called once during process execution lifetime when the script subsystem is unloaded.
   - `BeforeDocumentExtracted()`: Is called once per document in the Extraction module and fires before the OCR part of Extraction (as opposed to `DocumentExtracted()`, which fires after the OCR part of Extraction).

   Example:

```
namespace Custom.InputAccel.UimScript
{
    using Emc.InputAccel.UimScript;

    /// <summary>
    /// The base class for script assemblies.
    /// </summary>
    /// The script module must implement a class named ScriptMain that extends
    /// this class. Otherwise the module will not be loaded.
    public class ScriptMain : UimScriptMainCore
    {
        public ScriptMain()
            : base()
        {
        }

        /// <summary>
        /// Called when the script subsystem is first loaded before any other
events are fired.
        /// </summary>
        /// The implementing class can override this method to provide additional
        /// initialization. An overridden implementation must always call base class
        /// method before doing anything else.

        public override void ScriptLoad()
```

```
        {
            base.ScriptLoad();

            // Add your initialization here
        }

        /// <summary>
        /// Called when the script subsystem is unloaded.
        /// </summary>
        /// The implementing class can override this implementation to provide
        /// additional unload-related clean up. An overridden implementation must
always
        /// call the base class after doing everything else.
        public override void ScriptUnload()
        {
            // Add your unload code here

            base.ScriptUnload();
        }
    }
}
```

5. For each of the following, create a class and provide implementation and additional methods as required:

   - Document Type

     For examples, see "Document Type Scripts" on page 42.

   - Super Filter

     For an example, see "Super Filter Class Template Example" on page 80.

   - Batch Creation

     For an example, see "Batch Creation Class Template Example" on page 109.

6. Build the project and test by either deploying the script or debugging the script.

2.3.2 Deploying Profile Scripts

To deploy profile scripts:

1. Create the profile script in Visual Studio.
2. Create a `\bin` folder under the `<CaptureSystem>` directory if it does not exist.
3. Copy the DLL and data files to `<CaptureSystem>\bin`.
4. Copy any additional data files to the same folder.
5. Enter the names of the DLL and all data files as a comma-separated list in the **DeploymentFiles** global option for the **Capture System** in Intelligent Capture Designer.
6. Deploy the files to the server from Intelligent Capture Designer.

   Alternatively, you can use the command line utility, `<path-to-client-install>\Client\binnt\DeploymentUtility.exe` to deploy the files. To get help on how to run the utility, execute the following command:

```
deploymentutility.exe -help
```

Notes

- Make sure that you've already compiled any *XPPs*; this utility does not compile them.
- The maximum length of the command line is limited by Windows.
- Unicode characters are supported for solution names, profile names, and for the arguments file.

7. Restart the client modules.

2.3.3 Debugging Profile Scripts

To debug profile scripts:

1. Create and compile the profile script in Visual Studio.
2. Copy all DLL files (primary and secondary DLLs) and data files to `%AllUsersProfile%\EMC\InputAccel\Custom\DebugBin`.
3. Create and deploy a CaptureFlow that uses the script profile.
4. Create a batch with the CaptureFlow and process it until it is ready in the module to debug.
5. Configure Visual Studio to launch the module when debugging the DLL and set breakpoints as needed in the source.
6. Select **Start Debugging** in Visual Studio and then process the test batch.

2.4 Task Event Type Script Reference

Task event types are fired when the structure of the task or the type for a single document is changed. Events are delivered using public virtual methods. Event handling is optional and implemented by overriding the virtual event.

For more information about profile script class and DLL structure, see "Architecture" on page 27.

Table 2-2: Task Event Types

Event Name	Modules that Fire the Event	Description
"BeforeShowDocumentList" on page 35	Identification	The list of document types is about to be presented to the operator. Used to filter or order the list. **Note:** This event is new as of 7.5 and can be implemented only in task events; that is, it cannot be implemented in ScriptMain.
"BeforeShowTemplates" on page 35	Identification	The list of page templates is about to be presented to the operator. Used to filter or order the list. **Note:** This event is new as of 7.5 and can be implemented only in task events; that is, it cannot be implemented in ScriptMain.
"BeforeTaskFinish" on page 36	Completion Identification	The user has requested to finish a task. **Note:** This event is new as of 7.5 and can be implemented only in task events; that is, it cannot be implemented in ScriptMain.
"DocumentIdentified" on page 37	Identification	The type of a document has been assigned in Identification. **Note:** This event is new as of 7.5 and can be implemented only in task events; that is, it cannot be implemented in ScriptMain.

Event Name	Modules that Fire the Event	Description
"DocumentTypeChanged" on page 38	Completion	The Document Type has changed. **Note:** Previous to 7.5, this event was added to the ScriptMain class. To maintain backward compatibility, if this task event is not implemented, then the corresponding implementation in ScriptMain is still called; however, new code should use a task event script.
"FlagsChanged" on page 38	Completion Identification	Document, page, or field flags have been changed. **Note:** This event is new as of 7.5 and can be implemented only in task events; that is, it cannot be implemented in ScriptMain.
"NodeAdded" on page 39	Completion Identification	A node has been added to the task. **Note:** Previous to 7.5, this event was added to the ScriptMain class. To maintain backward compatibility, if this task event is not implemented, then the corresponding implementation in ScriptMain is still called; however, new code should use a task event script.

Event Name	Modules that Fire the Event	Description
"NodeDeleted" on page 39	Completion Identification	A node has been deleted from the task. **Note:** Previous to 7.5, this event was added to the ScriptMain class. To maintain backward compatibility, if this task event is not implemented, then the corresponding implementation in ScriptMain is still called; however, new code should use a task event script.
"NodeMoved" on page 40	Completion Identification	A node is moved either to a different parent node or to a new position in the same parent node. **Note:** Previous to 7.5, this event was added to the ScriptMain class. To maintain backward compatibility, if this task event is not implemented, then the corresponding implementation in ScriptMain is still called; however, new code should use a task event script.
"PageIdentified" on page 41	Identification	The template for a page has changed. **Note:** This event is new as of 7.5 and can be implemented only in task events; that is, it cannot be implemented in ScriptMain.

Event Name	Modules that Fire the Event	Description
"TaskLoad" on page 41	Completion Extraction Identification	The task has loaded.  **Note:** This event is new as of 7.5 and can be implemented only in task events; that is, it cannot be implemented in ScriptMain.

2.4.1 BeforeShowDocumentList

Syntax

```
C#: public virtual void BeforeShowDocumentList(IUimNodeData document,
IList<string> documentTypes);
```

```
VB.NET: Public Overridable Sub BeforeShowDocumentList(document As IUimNodeData,
documentTypes As IList(Of String))
```

Description

This event is fired before the list of document types is displayed to the operator.

Modules

This event is fired by the following module:

- Identification

Use Cases

This event might be used as follows:

- To change the document types that the operator can select based on other templates used in the task.
- To sort the list of document types to position the most applicable ones on top.

2.4.2 BeforeShowTemplates

Syntax

```
C#: public virtual void BeforeShowTemplates(UImNodeData page,
IList<IUimTemplate> nextPageTemplates);
```

```
VB.NET: Public Overridable Sub BeforeShowTemplates(page As IUimNodeData,
nextPageTemplates As IList(Of IUimTemplate))
```

Description

This event is fired before the list of templates for a page is shown to the operator.

Modules

This event is fired by the following module:

- Identification

Note: Templates are not used in information extraction projects. Thus, this event does not apply to information extraction projects.

Use Cases

This event might be used as follows:

- To change the templates that the operator can select based on other templates used in the task.
- To sort the list of templates to position the most applicable ones on top.

Note: Make sure keep at least one template for the operator to choose or have a way to exit the task without completing it.

2.4.3 BeforeTaskFinish

Syntax

```
C#: public virtual TaskFinishAction BeforeTaskFinish(IUimNodeData taskNode,
CloseReasonCode reasonCode);
```

```
VB.NET: Public Overridable Function BeforeTaskFinish(taskNode As IUimNodeData,
reasonCode As CloseReasonCode) As TaskFinishAction
```

Description

This event is fired when the task is about to complete.

Modules

This event is fired by the following modules:

- Completion
- Extraction
- Identification

Trigger Conditions

This event fires when any of the following occurs:

- The operator request to finish the task.
- The task auto-completes when auto-task advance is enabled.
- The task is being closed because the idle session timeout limit has been reached.
- All documents of the task are processed by Extraction.

Use Cases

This event might be used as follows:

- To prevent the user from exiting the task due to content or structural problems.
- To finish custom auditing for the task.

Remarks

The script author is responsible for ensuring that operators are not locked into a task that cannot be completed. Setting any result of `BeforeTaskFinish` in the Extraction module does not affect logic.

2.4.4 DocumentIdentified

Syntax

```
C#: public virtual void DocumentIdentified(IUimNodeData document,
IRecognitionProject project, IUimTemplate[] pageTemplates);
```

```
VB.NET: Public Overridable Sub DocumentIdentified(document As IUimNodeData,
project As IRecognitionProject, pageTemplates As IUimTemplate())
```

Description

This event is fired after the type of a document has changed in Identification when the step is configured to auto-assign templates.

Modules

This event is fired by the Identification module.

Note: Templates are not used in information extraction projects. Thus, this event does not apply to information extraction projects.

Use Cases

This event might be used as auto-apply templates to pages in the document based on its type.

Remarks

If the step is not configured to auto-assign templates on a document type change, then this event is not fired.

2.4.5 DocumentTypeChanged

Syntax

```
C#: public virtual void DocumentTypeChanged(IUimDataContext oldDoc,
IUimDataContext newDoc);
```

```
VB.NET: Public Overridable Sub DocumentTypeChanged(oldDoc As IUimDataContext,
newDoc As IUimDataContext)
```

Modules

This event is fired by the Completion module.

Use Cases

This event may be used to copy values from the old document to the new one, an action which is not otherwise done.

Remarks

The old document context is discarded after the event, so any changes to it are lost and no further events are fired for that document context.

2.4.6 FlagsChanged

Syntax

```
C#: public virtual void FlagsChanged(IUimDataContext document,
IUimDataContext pageData, IUimPageContext page,
IUimFieldDataContext field);
```

```
VB.NET: Public Overridable Sub FlagsChanged(document As IUimDataContext,
pageData As IUimDataContext, page As IUimPageContext,
field As IUimFieldDataContext)
```

Description

This event is fired when document, page, or field flags have changed.

Modules

This event is fired by the following modules:

- Completion
- Identification

Use Cases

This event might be used as follows:

- To change task data based on the applied flags.
- To change whether a task can be finished based on the applied flags.

2.4.7 NodeAdded

Syntax

```
C#: public virtual void NodeAdded(IUimNodeData node, IUimNodeData parent,
GlobalEventReason reason);
```

```
VB.NET: Public Overridable Sub NodeAdded(node As IUimNodeData, parent As IUimNodeData,
reason As GlobalEventReason)
```

Description

This event is fired after a node has been added to the task.

Modules

This event is fired by the following modules:

- Completion
- Identification

Use Cases

This event may be used to update values in one or more documents based on the structure of the task.

Triggering Conditions

This event fires when any of the following are true:

- The operator has added a new node to the task.
- The operator has split a node so that a new node is created.

2.4.8 NodeDeleted

Syntax

```
C#: public virtual void NodeDeleted(IUimNodeData node, IUimNodeData parent,
GlobalEventReason reason);
```

```
VB.NET: Public Overridable Sub NodeDeleted(node As IUimNodeData, parent As IUimNodeData,
reason As GlobalEventReason)
```

Description

This event is fired after a node has been deleted from the task.

Modules

This event is fired by the following modules:

- Completion
- Identification

Use Cases

This event may be used to:

- Update values in one or more documents based on the structure of the task.
- Capture information from a document before it is deleted.

Triggering Conditions

This event fires when any of the following are true:

- The operator has added a deleted node from the task.
- The operator has merged two nodes so that a node is deleted.

Remarks

Deleted node data is discarded after this event is complete and any changes made to the Documents in the deleted node are lost.

2.4.9 NodeMoved

Syntax

```
C#: public virtual void NodeMoved(IUimNodeData node, IUimNodeData newParent,
int newIndex, IUimNodeData oldParent, int oldIndex, GlobalEventReason reason);
```

```
VB.NET: Public Overridable Sub NodeMoved(node As IUimNodeData, newParent As
IUimNodeData,
newIndex As Integer, oldParent As IUimNodeData, oldIndex As Integer,
reason As GlobalEventReason)
```

Description

This event is fired after a node has been moved either between parents or to a new position in the same parent.

Modules

This event is fired by the following modules:

- Completion
- Identification

Use Cases

This event may be used to update values in one or more documents based on the structure of the task.

Triggering Conditions

This event fires when the operator has moved a node to a new position in the task.

Remarks

If the node has been moved to a new position in the same parent, the two parent values are the same. In all cases, the indexes give the old and new positions of the node in its parent.

2.4.10 PageIdentified

Syntax

```
C#: public virtual void PageIdentified(IUimNodeData page,
IRecognitionProject project, IUimTemplate[] pageTemplates);
```

```
VB.NET: public Overridable Sub PageIdentified(page As IUimNodeData,
project As IRecognitionProject, pageTemplates As IUimTemplate())
```

Description

This event is fired after and operator has changed the template for a page.

Modules

This event is fired by the Identification module.

Note: Templates are not used in information extraction projects. Thus, this event does not apply to information extraction projects.

Use Cases

This event might be used as follows:

- To auto-apply templates to other pages based on the current template.
- Initialize pre-index data for the new template.

2.4.11 TaskLoad

Syntax

```
C#: public virtual void TaskLoad(IUimNodeData taskNode);
```

```
VB.NET: Public Overridable Sub TaskLoad(taskNode As IUimNodeData)
```

Description

This event is fired after all documents have loaded but before the task is presented to the operator.

Modules

This event is fired by the following modules:

- Completion
- Extraction

- Identification

User Cases

This event might be used as follows:

- To initialize custom extraction logic used by the task.
- To start custom auditing for the task.

2.5 Document Type Scripts

Document Type scripts are contained in .NET DLLs with one class for each Document Type. You must create event handlers in the class to process both non-UI events such as validation rule execution and UI events such as button clicks on the data entry form. These events can read or manipulate the document data or the form itself. Document Type events run as part of a task and the context of document events is always the document to which the field or page belongs in the step that is currently executing. For example, if the task is at the folder level, multiple contexts will be created, one for each document in the folder.

The Document Type scripting model maintains a strict separation between a document object and a data entry form object. Attended client modules, such as Completion, have a data entry form object associated with a document object while unattended client modules, such as Extraction, only have a document object. A form object is always associated with a document object, but a document object may fire events even when it is not associated with a form object. In addition to document-level events, task-level scripting events are fired when the structure of the task changes—nodes added, deleted, or moved—and when the document type changes. In these cases, since the event occurs outside a single document, its scope is not limited to that of a single document.

Document Type scripts are written for the Completion, Identification, Extraction modules and Intelligent Capture Web Client.

For more information about using document type scripting in Intelligent Capture Web Client, see "Intelligent Capture Web Client Server-side Document Type Scripting" on page 127.

For more information about profile script class and DLL structure, see "Architecture" on page 27.

2.5.1 Completion

Document-type scripting in the Completion module consists of a set of events that are fired and objects that can be manipulated during those events through a large number of APIs. In many cases, the goal of an event handler is straightforward and might use only one or two APIs. For example, to reformat a telephone number, a script responds to the operator leaving the telephone number field by reading the value, reformatting it, and assigning the new value back to the field.

In other cases, multiple event handlers and multiple APIs can be combined to produce a composite behavior that is more significant than that of its individual parts. A comprehensive review of all such approaches is impossible: the number of variations and uses of scripting is simply too large. Instead, several examples are provided to survey some of the types of composition that can be accomplished with the available APIs. You can use these as both inspiration and starting points for your own application-specific scripted customizations.

2.5.1.1 Conditional Field Properties

Business Case

Field properties in a document type define behaviors that the field must always obey. For example, a field that indicates a percentage score may be configured to have a value range of 0–100. Completion would then never allow any value outside this range. Because the property is considered to be inherent to the field, it cannot be overridden at runtime. In complex documents, however, one or more properties may be conditional depending on the values of other fields. Instead of using the design-time property, conditional behavior can be achieved through a mix of validation rules and scripting.

Scenario

A requisition includes a list of requisitioned items with the following fields:

- The name of each requisitioned item (the field name in Intelligent Capture Designer is `ItemName`)
- The amount of each requisitioned item (`ItemAmount`)
- The approver's name (`Approver`)

When the Completion operator receives a task for this form, the name and amount of each requisitioned item is required. Furthermore, if any single item has an amount of more than $1,000.00, the approver's name is required; otherwise the approver's name can be empty.

Solutions

The `required` field property, when set, indicates that the field is invalid if its value is blank. As with all field properties, enabling the `required` property defines an inherent behavior that must always be true and cannot be overridden by scripting at

runtime. In this scenario, the item name and amount should be configured as required; the form is never valid without these fields. The approver's name, on the other hand, is only required when a high-value item has been requisitioned. Because the approver's name is conditionally-required, the `required` property should not be set and the behavior achieved through other means.

In addition to controlling the `required` state of the approver's name field, it may also be helpful to disable the field no item is of high value to avoid confusing the operator. This can be done by adding the following to the script:

```
public void ExitControlItemAmount(IUimFormControlContext control)
{
     IUimDataContext document = control.FieldDataContext.UimDataContext;
     bool needsApprover = HasHighValueItems(document);
     IUimFormControlContext approverName =
control.ParentForm.FindDataBoundControl("Approver");
     approverName.EnableControl(needsApprover);
}
```

Option 1

In this solution, conditionally-required behavior is achieved by using a validation rule that is disabled at runtime whenever the item amounts indicates that no items are of high value. To create the validation rule, in the document-type definition in Intelligent Capture Designer create a rule called `ApproverRequired` with the following properties:

Note: The list of dependent fields is read-only and is auto-populated when the rule definition is required.

Table 2-3: ApproverRequired Validation Rule Properties

Rule Property	Value
Name	ApproverRequired
Error Message	Required: Field must not be empty
Scope	Document
Method	Expression
Definition	`Length(Approver) > 0`
Dependent Fields	Approver

With this rule, the operator is shown a similar error whenever the approver's name field is empty. The second part of the solution is to dynamically enable and disable this rule. This must be done in two places: when the document first loads to ensure the initial state is correct and then any time an item amount changes. The code to accomplish this is as follows:

```
public class ScriptRequisition : UimScriptDocument
{
     // Initialize the rule when the document loads.
     public void DocumentLoad(IUimDataContext document)
     {
```

```
          RequireApprover(document);
     }

     // Update the rule state when leaving any ItemAmount control.
     // Note: If the approver's name field will be disabled, the code from the previous
     // section should be placed in this method.
     public void ExitControlItemAmount(IUimFormControlContext control)
     {
          IUimDataContext document = control.FieldDataContext.UimDataContext;
          RequireApprover(document);
     }

     // Determine whether any of the items is of high value
     private bool HasHighValueItems(IUimDataContext document)
     {
          IUimFieldDataContext[] itemAmounts = document.
FindArrayFieldDataContext("ItemAmount");
          for (int i=0; i<itemAmounts.Length; i++)
          {
               if (itemAmounts[i].ValueAsDecimal > 1000m)
               {
                    return true;
               }
          }
          return false;
     }

     // Enable or disables the rule based on the presence of high-value items.
     // If the rule state is already correct, the method will have no effect, but this is
     // handled by the scripting infrastructure and does not need to be managed by this
script.
     private void RequireApprover(IUimDataContext document)
     {
          bool needsApprover = HasHighValueItems(document);
          document.EnableValidationRule("ApproverRequired", needsApprover);
     }
}
```

Option 2

In this solution, conditionally-required behavior is achieved by using a validation rule that takes the item values into account. To create the validation rule, in the document-type definition in Intelligent Capture Designer create a rule called `ApproverRequired` with the following properties:

Note: The definition is blank for all scripted rules.

Table 2-4: ApproverRequired Validation Rule Properties

Rule Property	Value
Name	ApproverRequired
Error Message	Required: Field must not be empty
Scope	Document
Method	Expression
Definition	
Dependent Fields	Approver, ItemAmount

This rule will fail only when (1) one or more items is of high value and (2) the approver's name is empty. Because both fields affect the outcome of the rule, both are set as dependent fields for the rule. The script engine then ensures that the rule is reevaluated whenever either field changes. The rule script is as follows.

```
public class ScriptRequisition : UimScriptDocument
{
     // Pass or fail the rule based on the presence of high-value items and the
approver's name.
     private void ExecuteValidationRuleApproverRequired(IUimDataContext document)
     {
          bool needsApprover = HasHighValueItems(document);
          IUimFieldDataContext approverName =
document.FindFieldDataContext("ApproverName");
          if (needsApprover && string.IsNullOrEmpty(approverName.ValueAsString))
          {
                    // fail the rule
               throw new Exception();
          }

          // pass the rule by simply returning
     }

     // Determine whether any of the items is of high value
     private bool HasHighValueItems(IUimDataContext document)
     {
          IUimFieldDataContext[] itemAmounts = document.
FindArrayFieldDataContext("ItemAmount");
          for (int i=0; i<itemAmounts.Length; i++)
          {
               if (itemAmounts[i].ValueAsDecimal > 1000m)
               {
                    return true;
               }
          }

          return false;
     }
}
```

2.5.1.2 Conditional Requirement of Complete Tasks

Business Case

Completion is designed to allow operators to mark a task as complete at any time, even if it contains validation errors. This ensures that if the operator is unable to complete the work—such as when necessary data is illegible or the end of the shift has been reached—then the work is not lost as it would be if the operator cancelled the task. The CaptureFlow can then send incomplete tasks back to another operator, possibly rerouting to a different part of the workflow if the work was incomplete due to unfixable errors.

The business process, however, may instead require that operators either finish or discard the entire task. For example, when changes to the data are audited it is often easier to be able attribute the entire document to one operator than to manage which fields were edited by which operator. Scripting provides a property that can be set to use this behavior instead. In this case, the business may still need to provide the operator with a way to reject documents that cannot be completed.

Scenario

When the Completion operator receives a task for a claim form, the operator must complete all fields before completing the task and the module must enforce this. If the document cannot be completed due to missing pages or invalid data on the form, the opertor will check a box (called Rejected) indicating that the document is rejected after which the operator can complete the task without correcting all form errors.

If the task is above the document level, then when the operator rejects one document, the operator must still either correct the errors in other documents or reject those documents as well because the rejected state is tracked on a document-by-document basis.

Solution

First configure the **Rejected** checkbox to have the value `true` when checked and `false` when unchecked; the script will use these values to determine whether the document was rejected. The script for this scenario is responsible for two tasks: initially disabling the ability to complete tasks with errors (provided that the box is not checked) and subsequently reinstating that ability for rejected documents.

```
public class ScriptClaim : UimScriptDocument
{
     // Initialize the ability to finish with errors when the document loads.
     public void DocumentLoad(IUimDataContext document)
     {
          AllowIncompleteFinish(document);
     }

     // Update the state when leaving the Rejected control.
     public void ExitControlRejected(IUimFormControlContext control)
     {
          IUimDataContext document = control.FieldDataContext.UimDataContext;
          AllowIncompleteFinish(document);
     }

     // Enable or disables the ability to finish with errors based on the rejected state.
     private void AllowIncompleteFinish(IUimDataContext document)
     {
          IUimFieldDataContext rejected = document.FindFieldDataContext("Rejected");
          document.TaskFinishOnErrorNotAllowed = rejected.ValueAsBoolean;
     }
}
```

If the workflow supports multiple types of rejections by routing each type differently, then modify this solution to use a dropdown list of allowed options in place of a checkbox. In addition, modify the script to allow the task to finish when any of the non-rejected options are selected.

This approach controls whether the operator can complete a task with errors after rejecting a document. However, it does not automatically complete the task once the document has been rejected. Tasks cannot be programmatically completed though scripting.

2.5.1.3 Controlling Field Correction Level

Business Case

After extraction, low-confidence characters (such as those with the value ?) are marked in each field. When a field containing low-confidence characters is processed in Completion, the operator first enters the correct character for each after which the operator can correct field-level errors depending on the work level of the step. If the field contains many invalid characters, it is often easier to retype the entire field than it is to correct each character individually. The operator can clear the entire field to skip the bad characters by pressing the *clear field* keystroke; the default key on U.S. keyboards is ` (backtick).

When it is likely that the operator will frequently clear the field, the business could improve operator efficiency by automatically pre-processing the fields containing low-confidence characters. This could be applied when the number of low-confidence characters is high, when the percentage of the characters in the field that have low-confidence is high, or a mix of both. In these cases, the field could be cleared or the low-confidence characters could be accepted so that the operator corrects the entire field at once.

Scenario

Proof-of-income forms are received by fax tend to have a higher incidence rate of low-confidence characters due to the lower resolution and increased image skew. To avoid excessive character repair or field clearing, character repair will be used whenever the number of low-confidence characters exceeds the smaller of 5 characters or 20% of the field length. For example, a 5-digit ZIP code with 2 low-confidence characters would need to be retyped as low-confidence characters represent 40% of the field length, but a 40-character description with 6 low-confidence characters would be corrected character-by-character as it represents only 15% of the field length. To give the operator the decision over whether to correct or retype, the low-confidence characters will be accepted rather than clearing fields that exceed the threshold.

Solution

The sequence for accepting the low-confidence characters in a field while leaving the text in place is as follows:

1. The text is read from the field.
2. The value is assigned back to the field while clearing the low-confidence characters.
3. The original text is assigned back to the field.

The third step is necessary because, for number and date fields, setting the value to a string that cannot be parsed as a date or number will result in the field containing the default value. For example, if a script sets the value for a number field to `1?`, then the data will contain `0` and the form will show `0`. Replacing the field text can only be

done by updating the control in the data entry form; the field data only allows the value to be assigned. Because the form control is required, the correct place to auto-accept the low-confidence characters is on form load. The script is as follows:

```
public class ScriptProofOfIncome : UimScriptDocument
{
     public void FormLoad(IUimDataEntryFormContext form)
     {
          // Get a list of all fields
          IUimFieldDataContext[] fields = form.UimDataContext.GetFieldDataContextArray();

          for (int i = 0; i < fields.Length; i++)
          {
               IUimFieldDataContext field = fields[i];

               // We only need consider field with at least one unaccepted "?"
               if (field.NeedsCharacterRepair)
               {
                    // For each field, get the text, calculate the allowed threshold,
and count
                    // how many "?" the text contains
                    string fieldText = field.ValueAsString;
                    int threshQuestion = Math.Min(5, fieldText.Length / 5);
                    int countQuestion = 0;
                    for (int c = 0; c < fieldText.Length; c++)
                    {
                         if (fieldText[c] == '?')
                         {
                              countQuestion++;
                          }
                    }

                    // If the field has too many unaccepted "?", auto-accept them
                    if (countQuestion > threshQuestion)
                    {
                         field.SetValue(field.Value, LowConfidenceCharacters.Clear);
                         IUimFormControlContext control =
form.FindDataBoundControl(field.Name);
                         control.SetText(fieldText);
                    }
               }
          }
     }
}
```

This method will run each time the form is loaded, but after the first run it will have no effect as no fields will remain with too many unaccepted low-confidence characters.

If the field text is to be cleared rather than preserved, the logic can be moved to the post-extraction run event to ensure that it only ever runs once. Although additional runs on form load have no effect, the performance is slightly better to avoid them when possible. In this case the form load event is unnecessary as the script no longer needs to access the control. In this case, the logic for clearing a field is simply:

```
field.SetValue(null, LowConfidenceCharacters.Clear);
```

In this solution, when the text in a form control is set to an invalid value for a field, the field is marked in error, ensuring that the operator will be stopped on it. Care must be taken with text fields as the presence of a ? alone may not be enough to consider the field invalid. One approach to ensuring that the operator will be stopped on it. Care must be taken with text fields as the presence of a ? alone may not be enough to consider the field invalid. One approach to ensuring that the

operator is stopped on text fields is to add validation rules for text fields to reject it if it contains question marks. This will work in all cases with no additional scripting, but it requires one rule per text field which may be a significant number.

A different approach is to configure each text field to always require confirmation. Then in the form load event, if the field does not contain too many low-confidence characters, programmatically confirm the field. With this approach, the operator will be stopped to confirm any text field in which the low-confidence characters were accepted. The corresponding logic for the above script would be as follows with differences in bold:

```
if (field.NeedsCharacterRepair)
{
     // For each field, get the text, calculate the allowed threshold, and count
     // how many "?" the text contains
     string fieldText = field.ValueAsString;
     int threshQuestion = Math.Min(5, fieldText.Length / 5);
     int countQuestion = 0;
     for (int c = 0; c < fieldText.Length; c++)
     {
          if (fieldText[c] == '?')
          {
               countQuestion++;
          }
     }

     // If the field has too many unaccepted "?", auto-accept them
     if (countQuestion > threshQuestion)
     {
          field.SetValue(field.Value, LowConfidenceCharacters.Clear);
          IUimFormControlContext control = form.FindDataBoundControl(field.Name);
          control.SetText(fieldText);
     }
    else
     {
          field.SetDataConfirmed(true);
     }
}
else
{
     field.SetDataConfirmed(true);
}
```

2.5.1.4 Efficient Field Defaults

Business Case

Extraction is not guaranteed to return values for all fields in a document. A field could be missing from the paper form, OCR could fail on the field, or free-form approaches might fail to find the field. The typical approach to handling missing fields is to require that the operator manually enter a value and to consider the document incomplete until a value is entered. In some cases, however, it may be more efficient to automatically assign a default value for the field. This can be particularly useful when cross-field checks of the document will produce an error for the operator to correct should the default be wrong.

Default values can also be used to allow the operator to leave fields blank when they have a common value and then assign that value before export. For example, if a form had a text box for a branch ID, then the operator could be instructed to enter a

value only if it was different than the most common branch. Scripting could then populate the ID with that of the most common branch any time the field was blank.

When a form contains a large number of fields that need default values, the scripting to check whether each field needs to have the default applied and then do so gets lengthy and involves much repetition. Furthermore, if the default changes the script must be updated, recompiled, retested, and redeployed—a potentially-lengthy process. A solution using custom field values in the form reduces code complexity and enables changing defaults by updating the only document type and not its script.

Scenario 1

An order form has, among others, columns in an `Items` table for the price (`Price`), quantity (`Quantity`), discount rate (`Discount`). In most orders, the quantity for each item is almost always 1 and the discount rate is almost always 0%. In this form it is safe to assume these values whenever OCR fails as the cross-field validation rules that check the total amount of the order will fail if either assumption is incorrect. It is not safe to assume anything about the price as it varies too widely. Because all of these fields are required, the defaults should be applied after OCR, before the operator receives the form as a task.

Solution 1

In the document type, configure the custom value property for each of the above fields as follows:

Table 2-5: Items Table Custom Values

Field Name	Custom Value
Price	
Quantity	1
Discount	0

To apply the defaults to the fields in the form, read each field in the `Items` table and, if the field is blank and has a configured default, apply the default to the field. The code to accomplish this functionality is as follows:

```
public class ScriptOrder : UimScriptDocument
{
     public void DocumentExtracted(IUimDataContext document)
     {
          // Read the names of the columns in the table
          IUimTableSectionContext items = document.FindTableSection("Items");
          string[] fieldNames = items.GetFieldNames();

          for (int row = 0; row < items.RowCount; row++)
          {
               foreach (string fieldName in fieldNames)
               {
                    // Update each field in each row if it (1) has a default and (2) is
empty
                    IUimFieldDataContext field = items.GetFieldAt(row, fieldName);
```

```
                    if (string.IsNullOrEmpty(field.ValueAsString) &&
                        !string.IsNullOrEmpty(field.CustomValue))
                    {
                        field.SetValue(field.CustomValue);
                    }
                }
            }
        }
}
```

When a populated quantity is processed, the value assignment will be skipped because the field already has a value. When a price is processed, the value assignment will be skipped because the field does not have a configured default. Examining all fields in the table—including those that have no default value—allows defaults to be configured by changing the document type rather than the scripting source. In a scenario such as this where only two fields have defaults and adding defaults for other fields is unlikely, it would be easier to check just those two, but using the custom value property still provides the ability to change the defaults without changing the scripting.

Scenario 2

The order form in the previous scenario has separate sections for the billing and shipping information with a checkbox that indicates that the billing information should be used for shipping as well. To assist the operator, the designer has used scripting to clear and disable the shipping fields when the box is checked. The export step will then use the billing fields in place of the shipping fields as appropriate.

Solution 2

Conditional logic in the export step could be used to select between the two sets of fields, but this produces a lot of repetition of the condition. On document unload, the billing fields can be copied to the shipping fields, but this too results in repetition of the same code for each field. In the latter case, the script would look similar to the following:

```
public class ScriptOrder : UimScriptDocument
{
     public void DocumentUnload(IUimDataContext document)
     {
          IUimFieldDataContext billField, shipField;
          if (document.FindFieldDataContext("BillingIsShipping").ValueAsBoolean)
          {
               billField = document.FindFieldDataContext("BillingName");
               shipField = document.FindFieldDataContext("ShippingName");
               shipField.SetValue(billField.ValueAsString);
               billField = document.FindFieldDataContext("BillingStreet");
               shipField = document.FindFieldDataContext("ShippingStreet");
               shipField.SetValue(billField.ValueAsString);
               billField = document.FindFieldDataContext("BillingCity");
               shipField = document.FindFieldDataContext("ShippingCity");
               shipField.SetValue(billField.ValueAsString);

               // and so forth
          }
     }
}
```

Instead, the field-level custom value will be used to define the billing field that is the equivalent to the shipping field. In the document type, place the billing fields in a section named **Billing** and the shipping fields in a section named **Shipping**. Then set the custom value property for each field as follows:

Table 2-6: Billing and Shipping Custom Values

Field Name	Custom Value
BillingName	
BillingStreet	
BillingCity	
ShippingName	BillingName
ShippingStreet	BillingStreet
ShippingCity	BillingCity

In the document unload event when the shipping fields are to be copied from the billing fields, the script will then read the custom value for each field in the **Shipping** section, find the billing field with that name, and copy the value. The code for this functionality is the following:

```
public class ScriptOrder : UimScriptDocument
{
     public void DocumentUnload(IUimDataContext document)
     {
          IUimFieldDataContext billField, shipField;
          if (document.FindFieldDataContext("BillingIsShipping").ValueAsBoolean)
          {
               IUimTableSectionContext shipping = document.FindTableSection("Shipping");
               foreach (string field in shipping.GetFieldNames())
               {
                    shipField = document.FindFieldDataContext(field);
                    billField = document.FindFieldDataContext(field.CustomValue);
                    if (billField != null)
                    {
                         shipField.SetValue(billField.ValueAsObject);
                    }
               }
          }
     }
}
```

With this approach, all of the values are copied without the repetition. The example uses only three fields, but with a full set of information which could easily use 10 fields, the savings is much more significant.

2.5.2 Example of a Document Type Class Template

```
namespace Custom.InputAccel.UimScript
{
    using Emc.InputAccel.UimScript;

    /// <summary>
    /// The custom script class for the document type
    /// <Document Type Name as defined in Intelligent Capture Designer>.
    ///</summary>
    public class Script<Document Type Name as defined in Intelligent Capture Designer> :
UimScriptDocument
    {
        ///<summary>
        ///
        /// Default constructor.
        /// </summary>
        public Script<Document Type Name as defined in Intelligent Capture Designer>()
        : base()
        {
        }

        /// <summary>
        /// Executes when the Document is first loaded for the task by the Completion
module,
        /// after all of the runtime setup is complete.
        /// </summary>
        /// <param name="dataContext">The context object for the document.</param>
        public void DocumentLoad(IUimDataContext dataContext)
        {
        }

        /// <summary>
        /// Executes when the Document is first loaded for the task by the Extraction
        /// module, after all of the runtime setup is complete.
        ///</summary>
        /// <param name="dataContext">The context object for the document.</param>
        public void DocumentUnload(IUimDataContext dataContext)
        {
        }

        /// <summary>
        /// Executes a named validation rule.
        /// </summary>
        /// <param name="dataContext">The context object for the document.</param>
        public void ExecuteValidationRule<rulename>(IUimDataContext dataContext)
        {
        }

        /// <summary>
        /// Executes when the data entry form is loaded.
        /// </summary>
        /// <param name="controlContext">The context object for the data entry form.</
param>
        public void FormLoad(IUimDataEntryFormContext formContext)
        {
        }

        /// <summary>
        /// Executes when a UI control gains focus.
        /// </summary>
        /// <param name="controlContext">The context object for the control.</param>
        public void EnterControl(IUimFormControlContext controlContext)
        {
        }

        /// <summary>
        /// Executes when a UI control loses focus.
        /// </summary>
```

```
        /// <param name="controlContext">The context object for the control.</param>
        public void ExitControl(IUimFormControlContext controlContext)
        {
        }

        /// <summary>
        /// Executes when the operator confirms the field value by pressing Enter.
        /// </summary>
        /// <param name="controlContext">The context object for the control.</param>
        public void ConfirmControl(IUimFormControlContext controlContext)
        {
        }


        /// <summary>
        /// Executes when a button is clicked.
        /// </summary>
        /// <param name="controlContext">The context object for the control.</param>
        public void ButtonClick(IUimFormControlContext controlContext)
        {
        }
    }
```

2.5.3 External Interfaces and Data Models

This section describes major external data objects, and the services, methods, and APIs that operate upon them.

2.5.3.1 Resource Management

Scripts often use shared resources such as database connections that are time-consuming to create and destroy. We recommend that these connections be persisted between tasks and documents for efficiency. The base class `UimScriptDocument` provides two methods to store and access these shared resources.

Table 2-7: Resource Management

Member	Description
`FindGlobalResource(String)`	Returns a named global resource value. If a resource with the given name is not set, then it returns `null`.
`SetGlobalResource(String, Object)`	Sets or updates a named resource in the global context. The resource is kept until application exit time. Setting the value to `null` removes the resource from global storage.

Application resources are not associated with a specific Document class as they may be used by multiple Document classes; for example, a custom connector to a back-end system may be used by all CaptureFlows, independent of the document being processed. In this situation, all Document Type scripts can share the resource by using the same name; ensuring this is the responsibility of the script author. Additionally, if the system will make multiple connections to different back-end systems, each much be stored with a unique name; this, too, is the responsibility of the script author.

Application resources could be used by multiple threads in a single process; therefore it is vital that their use is thread-safe. The script engine enforces this by serializing the execution of all scripting events.

2.5.3.2 Script Execution

Scripts are executed in the context of the thread. The runtime ensures that multiple events are not called on different threads at the same time for a given Document instance. The runtime also ensures that all Form events are called on the UI thread. However, the runtime does not ensure that the same thread is used for all Document events, nor that the same thread is used for all Document instances in a task.

2.5.3.3 Script Instance Lifetime

When Extraction or Completion receive a task, it creates an instance of the script class for each document in the task that has a script. It then uses that instance for all fired events. When the task is done, the script class instance is released.

2.5.3.3.1 Extraction: Order of Events

The execution order for non-UI events in the Extraction module is the following:

1. `TaskLoad` (All documents have been loaded, but the task has not been processed yet.)
2. `BeforeDocumentExtracted` (Document is about to be processed by Extraction.)
3. `ExecutePopulationRule`, `ExecuteTableRowPopulationRule` (Executes all population rules, including one-time rules, to ensure any data clean-up is completed before running `DocumentExtracted`.)
4. `DocumentExtracted` (Document instance for a given task has been loaded and populated by Extraction.)
5. `ExecutePopulationRule`, `ExecuteTableRowPopulationRule` (Executes only those population rules which are not one-time rules and that had a dependent field changed in the previous events.)
6. `ExecuteValidationRule`, `ExecuteTableRowValidationRule` (Execute a Document-level or table row-level validation rule.)
7. `BeforeTaskFinish` (The task is about to complete.)
8. `DocumentUnload` (Task has ended and the Document instance is finalized. No other events are fired after this event, even if field values change.)

2.5.3.3.2 Completion: Order of Events

When a data entry form is launched in Completion, the runtime generates both Document data events and form UI control events. The execution order for UI and non-UI events in Completion is the following:

1. `DocumentLoad` (Document instance for a given task is loaded by Completion.)
2. `ExecuteValidationRule`, `ExecuteTableRowValidationRule` (Executes once for each validation rule in each document.
3. `TaskLoad` (All documents have been loaded, but the task has not been presented to the operator yet.)
4. `FormLoad` (Data entry form is loaded just before the form is displayed for a given document. May be called more than once during the task.)
5. `EnterControl`, `ButtonClick`, `SelectionChange`, `ConfirmControl`, and `FlagsChanged` (UI flow causes these events.)
6. `ExecutePopulationRule`, `ExecuteTableRowPopulationRule` (Executes only those population rules which are not one-time rules and that had a dependent field changed.)
7. `ExecuteValidationRule`, `ExecuteTableRowValidationRule` (Executes when dependent fields changed.)
8. `ExitControl` (UI flow causes this event.)
9. `BeforeTaskFinish` (The task is about to complete.)
10. `DocumentClosing` (Once for each document; however, `BeforeTaskFinish` should be used instead of this event.)
11. `DocumentUnload` (Task has ended and the Document instance is finalized. No other events are fired after this event, even if field values change.)

Note: The Form lifetime is different from the Document lifetime. For a given task instance, the Document is loaded once and unloaded once. At any time, a Document is associated with only one Form instance. A Form is not guaranteed to be created in Completion until the form is shown to the user; documents which are never shown to the user may never have Forms created. When a form is associated with a Document, `FormLoad` and other UI related events are fired.

A Form may be detached from the Document and a new Form may then be attached to the Document during a single task. After detaching a Form instance, another Form instance may be associated with the Document and at this time, a new `FormLoad` event is fired. Note that there is no `FormUnload` event. It is important that the Script code not save any Form-related object references as script class variables. Also, scripts should not expect that a `FormLoad` will always be called. From a Form context, scripts can access a Document object, but the reverse is not true.

2.5.3.3.3 Identification: Order of Events

The execution order for UI tasks in Identification is the following:

1. `DocumentLoad` (Once for each document for flagging and once for each page for pre-indexing.)
2. `ExecuteValidationRule`, `ExecuteTableRowValidationRule` (Executes each validation rule in each document.)
3. `TaskLoad` (All documents have been loaded, but the task has not been presented to the operator yet.)
4. `BeforeShowTemplates` (When a page is selected in the batch.)
5. When a document or page is identified, the related events in their order are:
   - `ExecutePopulationRule`, `ExecuteTableRowPopulationRule` (Executes all population rules to ensure any data clean-up is completed before DocumentLoad.)
   - `DocumentLoad` (For each newly-identified page.)
   - `ExecutePopulationRule`, `ExecuteTableRowPopulationRule` (Executes only those population rules which are not one-time rules and that had a dependent field changed in the previous events.)
   - `ExecuteValidationRule`, `ExecuteTableRowValidationRule` (Executes once for each validation rule when any dependent field has changed.)
   - `DocumentIdentified`, `PageIdentified` (Executes as appropriate for the operator action.)
6. `FormLoad` (When a page is selected in the batch.)
7. `EnterControl`, `ButtonClick`, `SelectionChange`, `ConfirmControl`, `BeforeShowDocumentsList` (UI flow causes these events.)
8. `ExecutePopulationRule`, `ExecuteTableRowPopulationRule` (Executes population rules which are not one-time rules and that had a dependent field changed.)
9. `ExecuteValidationRule`, `ExecuteTableRowValidationRule` (Executes when dependent fields changed.)
10. `ExitControl` (UI flow causes this event.)
11. `BeforeTaskFinish` (The task is about to complete.)
12. `DocumentClosing` (Once for each document; however, `BeforeTaskFinish` should be used instead of this event.)
13. `DocumentUnload` (Task has ended and the Document instance is finalized. No other events are fired after this event, even if field values change.)

2.5.3.4 Defining the Document Type Script Class

For each document type that uses scripting, the DLL includes a class named `Script<DocTypeName>` that extends `Emc.InputAccel.UimScript.UimScriptDocument`. The implementation of this class must include a parameter-less constructor.

To determine the correct value for `<DocTypeName>`, replace each space in the name of the document type in Intelligent Capture Designer with an underscore. Some examples are:

Document Type Name in Intelligent Capture Designer	Scripting Class Name
Invoice	`ScriptInvoice`
Enrollment Form	`ScriptEnrollment_Form`

For more information about the script class and DLL structure, see "Overview" on page 27.

2.5.3.5 Context Objects Passed to Event Methods

This section summarizes the objects that are passed to Document script events. When an event method is invoked, it is passed a context object. This context object is used to access runtime data. All context objects are passed in as interfaces and these interfaces are defined in the `Emc.InputAccel.CaptureClient.dll` assembly with the namespace `Emc.InputAccel.UimScript`. The Scripting events receive the following types of context objects.

Table 2-8: Context Objects Passed to Event Methods

Context Object	Description
`IUimDataContext`	Document data including all index field data.
`IUimFieldDataContext`	Index field data for a single field instance. In the case of array fields, this object represents a single element instance in the array.
`IUimTableSectionContext`	Represents a collection of fields that form a logical section of the document. If the section represents a table, all fields in the section will be array fields.
`IUimDataEntryFormContext`	Data entry form and all form elements.
`IUimFormControlContext`	A single UI element. Each form control is named uniquely and these names may differ from index field names for data bound controls.

Context Object	Description
`IUimFormSectionContext`	Represents a section of fields in the form. If the section represents a table, it is either shown as a grid or sub-form array.

2.5.3.6 Document-level Script Event Reference

This section provides a high-level overview of the event methods that are fired when a document is processed. Script authors create event handlers by creating public methods on the class that follow the correct naming syntax and signature. These methods are automatically detected by the script engine; event handlers are not explicitly registered. For example, by creating the method `[DocumentLoad(IUimDataContext)]`, the script engine will execute the method when the `DocumentLoad` event is fired.

Event handling is optional. If a corresponding event method does not exist, then script code is not executed when the event occurs.

2.5.3.6.1 Document Events

This section details the document-level non-UI events that script authors can handle by creating methods in the Document Type class. The events described in this section run at the document level.

Table 2-9: Document Events

Event Name	Description
"DocumentClosing" on page 61	Executes just before the document is finalized. **Note:** This event does not apply to scripting when used with REST Services or Intelligent Capture Web Client.
"DocumentLoad" on page 62	Document instance for a given task is loaded by the Completion module. **Note:** This event does not apply to scripting when used with REST Services or Intelligent Capture Web Client.
"DocumentExtracted" on page 62	Document instance for a given task has been loaded and populated by Extraction. **Note:** This event does not apply to scripting when used with REST Services or Intelligent Capture Web Client.

Event Name	Description
"DocumentUnload" on page 63	Executed when the task is done, either because it completed successfully, failed, or was canceled. **Note:** This event does not apply to scripting when used with REST Services or Intelligent Capture Web Client.
"ExecutePopulationRule" on page 63	Execute a document-level named population rule. Called as needed. Initially called after document extraction and later whenever any dependent field changes.
ExecuteValidationRule	Execute a document-level named validation rule.
"ExecuteTableRowPopulationRule" on page 64	Execute a row-level named population rule. Called as needed. Initially called after document extraction and later whenever any dependent field changes.
ExecuteTableRowValidationRule	Execute a row-level named validation rule.

DocumentClosing

Syntax

```
C#: public void DocumentClosing(IUimDataContext dataContext, CloseReasonCode reason);
```

```
VB.NET: Public Sub DocumentClosing(dataContext As IUimDataContext, reason As
CloseReasonCode)
```

Description

Executes just before the document is finalized. The reason code indicates to the script why the document is being closed; in turn, the script can use this reason code to implement reason-dependent behavior.

Use Cases

This event may be used as follows:

- Set output values for hidden fields.
- Free previously acquired resources.

Remarks

Because this event is the next to last one fired for a document in a given step, the following limitations apply:

- Validations are not performed for changes in the document data during this event.

- The document status IA value is not updated based on changes made during this event.
- Exceptions thrown by the script are caught but ignored; neither the script nor the operator will have further opportunity to interact with the document.

DocumentLoad

Syntax

```
C#: public void DocumentLoad(IUimDataContext dataContext);
```

```
VB.NET: Public Sub DocumentLoad(dataContext As IUimDataContext)
```

Description

Executes when the Document is first loaded for the task by the Completion module.

Use Cases

This event may be used to initialize the document-specific state, such as:

- Setting default values for fields.
- Acquiring and caching per-task resources.

DocumentExtracted

Syntax

```
C#: public void DocumentExtracted(IUimDataContext dataContext);
```

```
VB.NET: Public Sub DocumentExtracted(dataContext As IUimDataContext)
```

Description

This event is executed when the document has been created and populated for the task by the Extraction module.

Use Cases

This event may be used to initialize the document-specific state, such as:

- Setting default values for fields.
- Acquiring and caching per-task resources.

Remarks

The Extraction module loads the document and fires this event after all pages are processed and the extracted fields are combined into a single Document.

DocumentUnload

Syntax

```
C#: public void DocumentUnload(IUimDataContext dataContext);
```

```
VB.NET: Public Sub DocumentUnload(dataContext As IUimDataContext)
```

Description

This event is executed after the task is done, either because it completed successfully, failed, or was canceled.

Use Cases

This event may be used to:

- Set output values for hidden fields.
- Free previously acquired resources.

Remarks

This is the last event that is fired for a document in a given step. Validations are not performed for any changes to the document data during this event. The `DocStatus` IA value is not updated based on any changes made during this event. Any exceptions thrown by the script are ignored; neither the script nor the operator will have further opportunity to interact with the document.

ExecutePopulationRule

Syntax

```
C#: public void ExecutePopulationRule<Rule name>(IUimDataContext dataContext);
```

```
VB.NET: Public Sub ExecutePopulationRule<Rule name>(dataContext As IUimDataContext)
```

Description

This event is executed after the runtime has determined the need for script-based population.

Triggering Conditions

Script population rules are run after any of the following has occurred:

- The document is first extracted.
- In the current document, any dependent fields have changed.

Note: In this case, one-time population rules are not run.

Remarks

When a field changes, first population rules are run and then validation rules. During the execution of the script, no other script methods are called recursively by the runtime and events are ignored.

ExecuteValidationRule

Syntax

```
C#: public void ExecuteValidationRule<Rule name>(IUimDataContext dataContext);
```

```
VB.NET: Public Sub ExecuteValidationRule<rule name>(dataContext As IUimDataContext)
```

Description

This event is executed after the runtime has determined the need for script-based validation.

Triggering Conditions

Script validation rules are run after any of the following is true:

- Once after `DocumentLoad` but before `FormLoad`.
- Any dependent field has changed.

Remarks

Whenever possible, scripts should not modify data and should treat these methods as read-only.

To customize the validation error message, you must call `SetValidationRuleFailMessage()` on the document for the rule and throw an exception to indicate the rule failure.

ExecuteTableRowPopulationRule

Syntax

```
C#: public void ExecuteTableRowPopulationRule<rule name>(IUimTableSectionContext
tableSectionContext, int rowIndex);
```

```
VB.NET: Public Sub ExecuteTableRowPopulationRule<rule name>(tableSectionContext As
IUimTableSectionContext,

rowIndex As Integer)
```

Description

This event is executed after the runtime has determined the need for script-based population of a single table row.

Triggering Conditions

Script population rules are run after any of the following actions occurs:

- The document is first extracted.
- In the current document, any dependent fields have changed.

Note: In this case, one-time population rules are not run.

Remarks

When a field changes, first population rules are run and then validation rules. During the execution of the script, no other script methods are called recursively by the runtime and events are ignored.

ExecuteTableRowValidationRule

Syntax

```
C#: public void ExecuteTableRowValidationRule<rule name>(IUimTableSectionContext
tableSectionContext, int rowIndex);
```

```
VB.NET: Public Sub ExecuteTableRowValidationRule<rule name>(dataContext As
IUimTableSectionContext, rowIndex As Integer)
```

Description

This event is executed after the runtime has determined the need for script-based validation for a single table row.

Triggering Conditions

Script validation rules are run after any of the following are true:

- Once after `DocumentLoad` but before `FormLoad`.
- Any dependent fields have changed in the current row.

Remarks

Whenever possible, scripts should not modify data and should treat these methods as read-only.

To customize the validation error message, you must call `SetValidationRuleFailMessage()` on the document for the rule and throw an exception to indicate the rule failure.

2.5.3.6.2 DataEntry Form Events

This section details the document-level UI events. An event may be associated with a source control like a text box or a button, and in this case the script author may choose to trap an event for the specific control or for a generic event. For example, if a Form has multiple buttons, the script can implement either multiple event handlers (one for each button), a single, generic event handler for all of the buttons, or a mix where some buttons have button-specific event handlers and the rest share a generic event handler. The events described in this section only run in the Completion module in response to user actions to change the content of the document. These events are fired only during the lifetime of the Form associated with the document.

Table 2-10: DataEntry Form Events

Event Name	Description
"FormLoad" on page 66	Data entry form is loaded. May be called more than once during task lifetime. Use this method to recreate the state of the Form such as enabling, disabling, hiding, and showing controls.
EnterControl	The UI control is preparing to lose focus. This event is fired for controls of type: text box, list box, combo box, and checkbox.
ExitControl	The UI control is exited, losing focus. This event is fired for controls of type: text box, list box, combo box, and checkbox.
ConfirmControl	User confirms field contents. Fired for all editable controls.
ButtonClick	Button is clicked for user created buttons on the Form.
SelectionChange	Selection is changed for list box or combo box.
TextChanged	Text box text is changed. The event is fired immediately after any change to the text, even if the field is still in focus.

FormLoad

Syntax

```
C#: public void FormLoad(IUimDataEntryFormContext formContext);
```

```
VB.NET: Public Sub FormLoad(formContext As IUimDataEntryFormContext)
```

Description

This event is executed when a Form instance is associated with a document instance.

Use Cases

A script may use this event to initialize the User Interface (UI) to:

- Hide form sections or controls.
- Localize UI elements.

Triggering Conditions

This event is fired any time the application constructs a Form UI. Since the Form UI may be discarded when the operator switches between documents in the task, this event may fire multiple times for a given document.

Remarks

The event is not guaranteed to fire for all documents in a task. If the user does not display the Form for a document, the Form is not loaded. The event is guaranteed to fire before any other UI event for a given Form. There is no corresponding `FormUnload` event.

EnterControl

Syntax

```
C#: public void EnterControl(IUimFormControlContext controlContext);
```

```
VB.NET: Public Sub EnterControl(controlContext As IUimFormControlContext)
```

Syntax

```
C#: public void EnterControl<control name>(IUimFormControlContext
controlContext);
```

```
VB.NET: Public Sub EnterControl<control name>(controlContext As
IUimFormControlContext)
```

Description

This event is called when a UI control gains focus. The event is fired only for data bound text box, list box, combo box, and checkbox controls. Script authors may provide a control specific implementation by using a method name with control name prefix or implement a generic method for all controls. If a method with control name exists, then that method is invoked. If such a method does not exist but a generic method exists, then the generic method is invoked. Note that the control name is used in the method signature and not the index field name. In general **Document Type** designer assigns the **Index Field** name to bound UI controls but this is not always the case.

Use Cases

A script may use this event to do one or more of the following:

- Populate the field with a default value. For example, when an “End Date” field gets focus, the script could set its value to that of the “Start Date”.

- Capture the initial field value for custom auditing. For example, when an "Account Number" field gets focus, the script could store the initial value of the field to log the change to the field value when the task is complete.

Triggering Conditions

A control gets focus when any of the following occur:

- The Form has been loaded and focus is set to the first field.
- The operator uses the **TAB** or **ENTER** key to navigate to the field.
- The operator clicks to set focus on a specific field.
- A script programmatically sets focus to a field.

Remarks

`EnterControl` and `ExitControl` are not guaranteed to fire in matched pairs. See `ExitControl` for more information.

ExitControl

Syntax

```
C#: public void ExitControl(IUimFormControlContext controlContext);
```

```
VB.NET: Public Sub ExitControl(controlContext As IUimFormControlContext)
```

Syntax

```
C#: public void ExitControl<control name>(IUimFormControlContext
controlContext);
```

```
VB.NET: Public Sub ExitControl<controlname>(controlContext As
IUimFormControlContext)
```

Description

This event is called before a request to lose focus is processed for a UI control. The event is fired only for data bound text box, list box, combo box, and checkbox controls. Script authors may provide a control specific implementation by using a method name with the control name prefix or implement a generic method for all controls. If a method with control name exists then that method is invoked. If such a method does not exist but a generic method exists, then the generic method is invoked. Note that the control name is used in the method signature and not the index field name. In general **Document Type** designer assigns the **Index Field** name to bound UI controls but this is not always the case.

Use Cases

A script may use this event to do one or more of the following:

- Apply custom data parsing for the field value. For example, if an operator enters six digits with no separators into a date field, the script could parse the six-digit string and assign the appropriate date to the field.

- Auto-populate fields left blank. For example, if an operator leaves an empty date field, the script could populate the field with the current date.
- Populate other fields using the value for the current field. For example, after an operator enters a customer's account number, the script could look up and populate the customer's name and address from the database information and skip to the next empty field.
- Show/hide or enable/disable other Form controls based on the value for the current field. For example, after an operator enters a non-zero value for the number of dependents, the script could show a table for entering dependent information.

After the event fires, focus will be set as follows:

- If the script explicitly sets focus to another field by using `IUimFormControlContext.SetFocus` on the field, that field will get the focus. If the event is being fired as part of navigating to a different document or task, the focus change is ignored.
- If focus is not set by the script, but the field value changes when the operator presses the **ENTER** key:
  - Focus stays on the current field if the field has an error.
  - Focus advances to the next appropriate field if the field does not have an error.
- If focus is not set by the script, and either the field value has not changed or the field is losing focus for any reason other than the operator pressing the **ENTER** key, focus advances to the next appropriate field, depending on why the field is losing focus.

Triggering Conditions

A control receives a request to lose focus when any of the following occur. These actions all cause the method to fire, independent of whether the value in the field has changed.

- The operator presses **ENTER** to move to the next field that needs work.
- The operator presses **TAB** to move to the next field in the Form.
- The field loses focus in Windows.

Remarks

If the focus stays on the current field, `EnterControl` is not fired again. This means that *EnterControl* and `ExitControl` are not guaranteed to fire in matched pairs.

ConfirmControl

Syntax

```
C#: public void ConfirmControl(IUimFormControlContext controlContext);
```

```
VB.NET: Public Sub ConfirmControl(controlContext As IUimFormControlContext)
```

Syntax

```
C#: public void ConfirmControl<control name>(IUimFormControlContext
controlContext);
```

```
VB.NET: Public Sub ConfirmControl<control name>(controlContext As
IUimFormControlContext)
```

Description

This event is executed when the user confirms a field value by pressing the **ENTER** key. Script authors may provide a control specific implementation by using a method name with the control name prefix or implement a generic method for all controls. If a method with control name exists then that method is invoked. If such a method does not exist but a generic method exists, then the generic method is invoked. Note that the control name is used in the method signature and not the index field name. In general **Document Type** designer assigns the **Index Field** name to bound UI controls but this is not always the case.

Triggering Conditions

The event is fired on a field only when both of the following are true:

- The field was configured to require confirmation in Intelligent Capture Designer > Document Types.
- The field has not been confirmed.

Remarks

Once the user has confirmed the field value, the event will not be fired again for that field if the user enters the field and presses **ENTER** again.

ButtonClick

Syntax

```
C#: public void ButtonClick(IUimFormControlContext controlContext);
```

```
VB.NET: Public Sub ButtonClick(controlContext As IUimFormControlContext)
```

Syntax

```
C#: public void ButtonClick<control name>(IUimFormControlContext
controlContext);
```

```
VB.NET: Public Sub ButtonClick<control name>(controlContext As
IUimFormControlContext)
```

Description

This event is fired when a user-created button on a form is clicked either using a mouse button or with a shortcut key. Script authors may provide a control specific implementation by using a method name with the control name suffix, a generic

method for all controls, or a mix of the two. If a method with the control name exists, then that method is invoked. If such a method does not exist but a generic method exists, then the generic method is invoked. Note that the control name is used in the method signature and not the index field name. In general **Document Type** designer assigns the **Index Field** name to bound UI controls but this is not always the case. Intelligent Capture Designer shows the control name or all controls.

Use Cases

This event may be used to:

- Perform expensive validations on demand by an operator.
- Populate one set of fields from database lookups using values from another set.

Notes

Additionally, this event can be used in combination with the `IUimFormControlContext.ShowControl` method to provide only shortcut keys for document-specific operations. The `IUimFormControlContext.ShowControl` method is used to hide a button on the data-entry form. In turn, the button is called by the `ButtonClick` event.

For more information about the `IUimFormControlContext.ShowControl` method, see the *Emc.InputAccel.CaptureClient API Reference Guide*.

SelectionChange

Syntax

```
C#: public void SelectionChange(IUimFormControlContext controlContext);
```

```
VB.NET: Public Sub SelectionChange(controlContext As IUimFormControlContext)
```

Syntax

```
C#: public void SelectionChange<control name>(IUimFormControlContext
controlContext);
```

```
VB.NET: Public Sub SelectionChange<control name>(controlContext As
IUimFormControlContext)
```

Description

This event is fired when a list box or combo box selection change occurs. Script authors may provide a control specific implementation by using a method name with the control name suffix, a generic method for all controls, or a mix of the two. If a method with the control name exists, then that method is invoked. If such a method does not exist but a generic method exists, then the generic method is invoked. Note that the control name is used in the method signature and not the index field name. In general **Document Type** designer assigns the **Index Field** name to bound UI controls but this is not always the case. Intelligent Capture Designer shows the control name or all controls.

Use Cases

This event may be used to show or hide a set of Form fields based on the selection.

Triggering Conditions

This event fires any time the text changes when any of the following occur:

- The operator selects a new value.
- The value is changed by a script.

Remarks

The event is fired immediately after the selection changes, even if the field still has focus. The field value is immediately updated to the new selection.

TextChanged

Syntax

```
C#: public void TextChanged(IUimFormControlContext controlContext);
```

```
VB.NET: Public Sub TextChanged(controlContext As IUimFormControlContext)
```

Syntax

```
C#: public void TextChanged<control name>(IUimFormControlContext
controlContext);
```

```
VB.NET: Public Sub TextChanged<control name>(controlContext As
IUimFormControlContext)
```

Description

This event is fired when the text in a text box changes. Script authors may provide a control specific implementation by using a method name with the control name suffix, a generic method for all controls, or a mix of the two. If a method with the control name exists, then that method is invoked. If such a method does not exist but a generic method exists, then the generic method is invoked. Note that the control name is used in the method signature and not the index field name. In general **Document Type** designer assigns the **Index Field** name to bound UI controls but this is not always the case.

Triggering Conditions

This event fires any time the text changes when any of the following occur:

- The operator types in the field.
- The operator cuts or pastes text.
- The operator performs rubber-band OCR.
- The operator hits a hotkey to clear the field.
- The field value is changed by a script.

- The form text is changed by a script.

Remarks

The event is fired immediately after the text changes, even if the field still has focus. When rubber-band OCR or a script changes the field value, the control text is updated to the new value before the event is fired. Other changes to the control text do not cause the field value to update until the field loses focus.

2.5.3.6.3 Table Events

This section details the table-level non-UI events. The events described in this section run at the document level in response to changes in the table structure. However, you cannot use the `InsertNewRow`, `DeleteRow` and `ClearTable` methods in table events.

Although the events are the result of operator UI actions, these are data-only events and the Form context is not available.

Table 2-11:

Event Name	Description
"InsertRow" on page 73	A new table row is inserted.
"Delete Row" on page 74	A table row is about to be deleted. Changes to the row will be lost.
"RemoveAllRows" on page 74	All table rows are about to be deleted. The `DeleteRow` event is not fired for each row when all rows are removed as a single operation.
"MoveRow" on page 75	A row has been moved to a different position.
"TableExtracted" on page 75	A table extraction has been performed.
"MergeRows" on page 75	Data in two rows has been merged into one row.
"TableStructureChanged" on page 76	Table structure change has been completed.

InsertRow

Syntax

```
C#: public void InsertRow(IUimTableSectionContext tableContext, int rowIndex);
```

```
VB.NET: Public Sub InsertRow(tableContext As IUimTableSectionContext,
rowIndex As Integer)
```

Description

This event is fired after a row has been added to a table.

Use Cases

This event may be used to initialize columns in the newly-added row such as setting the default values.

Delete Row

Syntax

```
C#: public void DeleteRow(IUimTableSectionContext tableContext, int rowIndex);
```

```
VB.NET: Public Sub DeleteRow(tableContext As IUimTableSectionContext,
rowIndex As Integer)
```

Description

This event is fired immediately before a row is deleted from a table.

Use Cases

This event may be used to capture values from the row before it is deleted.

Remarks

If the user deletes several rows at once, this event fires once per row. Any changes made to the row during this event are lost and no further events are fired for that row or its fields.

RemoveAllRows

Syntax

```
C#: public void RemoveAllRows(IUimTableSectionContext tableContext);
```

```
VB.NET: Public Sub RemoveAllRows(tableContext As IUimTableSectionContext)
```

Description

This event is fired immediately before all rows are deleted from a table.

Use Cases

This event may be used to capture values from the rows before they are deleted.

Remarks

This is the only event that is fired when the operator deletes all rows from the table using the **Clear Table** menu item; individual `DeleteRow` events are not fired. Any changes made to the table rows during this event are lost and no further events are fired for the table rows or their fields.

MoveRow

Syntax

```
C#: public void MoveRow(IUimTableSectionContext tableContext, int rowIndex);
```

```
VB.NET: Public Sub MoveRow(tableContext As IUimTableSectionContext,
rowIndex as Integer)
```

Description

This event is fired after one or more table rows have been moved to a new position.

Use Cases

This event may be used to reprocess calculated fields in the table rows such as a row number or a running total.

Remarks

The row index is the original index of the first moved row.

TableExtracted

Syntax

```
C#: public void TableExtracted(IUimTableSectionContext tableContext,
int rowIndex);
```

```
VB.NET: Public Sub TableExtracted(tableContext As IUimTableSectionContext,
rowIndex As Integer)
```

Description

This event is fired after multiple rows have been added to a table using the **Table Extraction** mode.

Use Cases

This event may be used to initialize columns in the newly-added rows such as setting default values for fields that were not extracted.

Remarks

`InsertRow` events are not fired for the newly-added rows. The row index describes where the new rows were added.

MergeRows

Syntax

```
C#: public void MergeRows(IUimTableSectionContext tableContext, int rowIndex);
```

```
VB.NET: Public Sub MergeRows(tableContext As IUimTableSectionContext,
rowIndex As Integer)
```

Description

This event is fired immediately after a table row is merged with the previous row.

Use Cases

This event may be used to clean unwanted values from the rows before they are merged.

Remarks

The row index is that of the selected (second) row before the rows were merged.

TableStructureChanged

Syntax

```
C#: public void TableStructureChanged(IUimTableSectionContext tableContext);
```

```
VB.NET: Public Sub TableStructureChanged(tableContext As
IUimTableSectionContext)
```

Description

This event is fired after any table structure change has completed.

Use Cases

This event may be used to update fields such as row numbers or column totals that change after any event that updates the table.

Remarks

When the tables structure changes, this event fires after all other table events and after the change is complete. For events such as `DeleteRow` which are fired before the table data changes, the sequence is:

- `DeleteRow` fires
- The row is deleted
- `TableStructureChanged` fires

For events such as `InsertRow` which are fired after the table data changes, the sequence is:

- The new row is inserted
- `InsertRow` fires
- `TableStructureChanged` fires

2.5.3.6.4 Global Event Types

This section details the global events that do not run in the context of a single document. The following types of events are fired when the structure of the task or the type for a single document is changed. Event handlers are created by overriding the corresponding virtual methods in the ScriptMain-derived class. Event handling is optional. If a corresponding event method does not exist, then the event is not fired.

Table 2-12: Global Event Types

Event Name	Description
"BeforeDocumentExtracted" on page 77	The document is about to be processed by Extraction.
"DocumentTypeChanged" on page 38	The document Type has changed. **Note:** Deprecated as of 7.5. Instead you should override the virtual event. For more information, see "Task Event Type Script Reference" on page 31.
"ModuleBatchListView" on page 78	The list of batches is about to be displayed to the operator. Used to filter or order the list.
"NodeMoved" on page 40	A node is moved either to a different parent node or to a new position in the same parent node. **Note:** Deprecated as of 7.5. Instead you should override the virtual event. For more information, see "Task Event Type Script Reference" on page 31.
"NodeAdded" on page 39	A node has been added to the task. **Note:** Deprecated as of 7.5. Instead you should override the virtual event. For more information, see "Task Event Type Script Reference" on page 31.
"NodeDeleted" on page 39	A node has been deleted from the task. **Note:** Deprecated as of 7.5. Instead you should override the virtual event. For more information, see "Task Event Type Script Reference" on page 31.

BeforeDocumentExtracted

Syntax

```
C#: public virtual void BeforeDocumentExtracted(IDictionary<string, object>
values);
```

```
VB.NET: Public Overridable Sub BeforeDocumentExtracted(values As
IDictionary(Of String, Object))
```

Description

This event is fired before any data is extracted in the task.

Modules

This event is fired by the Extraction module.

Use Cases

This event might be used to initialize custom extraction logic used by free-form extraction scripts.

ModuleBatchListView

Syntax

```
C#: protected virtual void ModuleBatchListView(string moduleType,
List<ITableRow> tableRowList, string loginName, string[] departments)
```

```
VB.NET: Protected Overridable Sub ModuleBatchListView(moduleType As String,
tableRowList As List(Of ITableRow), loginName As String,
departments As String())
```

Description

This event is fired when the list of batches is about to be displayed to the operator in the task chooser.

Modules

This event is fired by the following modules:

- Completion
- Identification

Use Cases

This event might be used as follows:

- To filter the list of batches to prevent the same operator from keying a document twice in a double-key environment.
- To change the order in which the batches are displayed.

Remarks

- The script author is responsible for ensuring that batches are not excluded from all operators.

- For more information, see "Batch Filtering Scripts" on page 111 and "Batch Filtering Script Reference" on page 113.

2.6 Image Processing Scripts

Image Processing scripting provides additional control over how to apply a set of image processing filters (also known as super filters). Scripting can enable image processor profiles to dynamically change settings, conditionally skip or reorder filters, change or remove return values from individual filters, and add custom return values. Image Processing scripts are contained in .NET DLLs with one class per Super Filter. The script implements the method in that class that is executed when the Super Filter is applied. The script can then apply the individual filters in the Super Filter in any order, can skip filters, and can change the input and output values from each filter.

For more information about profile script class and DLL structure, see "Architecture" on page 27.

2.6.1 External Interfaces and Data Models

This section describes major external data objects, and the services, methods, and APIs that operate upon them.

2.6.1.1 Image Processing Script Execution

When a task is received, the runtime looks for a script class with the appropriate name and if one is found, it creates a new instance and executes the `ExecuteSuperFilter` method. There is no guarantee that the same thread is used for all image processing tasks. Script code must not expect thread affinity.

2.6.1.2 Defining the Super Filter Script Class

This section defines the structure of the class that script authors must create in the scripting DLL for each Super Filter. Refer to "Overview" on page 27 to learn the structure of the main class that script authors must create in the scripting DLL.

For each Super Filter that will use scripting, the DLL must include a class named `Script<SuperFilterName>` that extends `Emc.InputAccel.UimScript.UimScriptFilter`. The implementation of this class must include a parameter-less constructor.

To determine the correct value for `<SuperFilterName>`, replace each space in the name of the image processing profile in Intelligent Capture Designer with an underscore. Some examples are:

Super Filter Name in Intelligent Capture Designer	Scripting Class Name
Bitonal	`ScriptBitonal`

Super Filter Name in Intelligent Capture Designer	Scripting Class Name
Color Cleanup	`ScriptColor_Cleanup`

2.6.1.3 Context Objects Passed to Event Methods

When a scripted Super Filter is executed, it is passed a context object which is used to access runtime data. The context object is of type `IUimScriptFilterContext` and is defined in `Emc.InputAccel.CaptureClient.dll` assembly with a namespace `Emc.InputAccel.UimScript`.

2.6.1.4 Super Filter Class Template Example

```
namespace Custom.InputAccel.UimFilterScript
{
    using Emc.InputAccel.UimScript;

    /// <summary>
    /// Custom script class for superfilter <Superfilter Profile Name as defined in the
Image Processing
    /// </summary>
    public class Script<Superfilter Profile Name as defined in Intelligent Capture
Designer> : UimScriptFilter
    {
        /// <summary>
        /// Default constructor.
        /// </summary>
        public Script<Superfilter Profile Name as defined in Intelligent Capture
Designer>()
            : base()
        {
        }

        /// <summary>
        /// Executes a scripted image processing profile filter
        /// </summary>
        /// <remarks>
        /// <para>
        /// The <paramref name="outValues"/> are populated by the individual filters
        /// as they are applied to the image. The script can keep, modify, or remove
these
        /// values.
        /// </para>
        /// <para>
        /// Once the task is complete, any values in the dictionary that have the same
name
        /// and type as an MDF value for the step will be stored as IA values. User-
defined
        /// values can also be saved provided that the user has added those values to
either
        /// the MDF file or through Intelligent Capture Designer.
        /// </para>
        /// </remarks>
        /// <param name="imageData">The input image onto which filters are applied.</
param>
        /// <param name="stepCustomValue">The step-specific custom scripting value.</
param>
        /// <param name="uimScriptFilterContext">The runtime context for the filter.</
param>
        /// <param name="outValues">A collection of named values returned by the
method.</param>
        /// <returns>The modified image after filters have been applied.</returns>
        public ImageData ExecuteSuperFilter(ImageData imageData,
            string stepCustomValue,
```

```
            IUimScriptFilterContext uimScriptFilterContext,
            Dictionary<string, object> outValues)
        {
        }
    }
}
```

2.6.2 Image Processing Script Events Reference

This section details the only Image Processing method.

2.6.2.1 ExecuteSuperFilter

Syntax

```
C#: public abstract ImageData ExecuteSuperFilter(ImageData imageData,
string stepCustomValue,
IUimScriptFilterContext uimScriptFilterContext,
Dictionary<string, object> outValues);
VB.NET: Public MustOverride Function ExecuteSuperFilter(imageData As ImageData,
stepCustomValue As String, uimScriptFilterContext As IUimScriptFilterContext,
outValues As Dictionary(Of String, Object>) As ImageData
```

Description

Executed when Image Processor runs a scripted Super Filter.

Use Cases

This event might be used as follows:

- Modify the inputs to individual filters.
- Modify the outputs returned by individual filters.
- Conditionally skip or reorder the individual filters.

2.6.3 Image Processor Filters Scripting Reference

This section provides a detailed description of Image Processor filters and their properties that can be used while scripting.

2.6.3.1 Detection Filters

A set of detection filters, which verify the type of content on an image.

2.6.3.1.1 DetectBarcodes

A filter enabling detection of various types of barcodes; any number of barcodes can be detected on one page.

Note: Test the filter on images similar to those used in production mode.

In some cases you may need to perform additional image processing before applying the barcode filter. For example, use the following filters:

- Deskew: to straighten skewed images.
- Remove Noise: to clean up dirty images.
- Line Removal: to repair the lines in the barcode itself.
- Smooth Edges: to smooth the lines in the barcode itself.

Table 2-13: DetectBarcodes Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Decode	Gets or sets a value that indicates whether barcode results should be decoded into readable strings.	bool	
Mode	Gets or sets the barcode detection mode.	string	• **Enhanced**: Provides better results by performing additional image preprocessing. The analyzing process is slower. • **Normal**: Enables quick barcodes detection.
Orientation	Gets or sets the orientation(s) of the barcodes to be detected.	string	• **Horizontal** • **HorizontalVertical** • **HorizontalVerticalDiagonal** • **Vertical**

Property name (non-translated programmatic name)	Description	Type	Values
ScanDistance1D	Gets or sets the scan distance, in pixels, between line sweeps when searching for 1D barcodes. Reducing the value can help in finding barcodes which are short relative to their height.	integer	1 to 10
BarcodeTypes	Gets or sets the type(s) of barcodes to be detected.	Comma separated string of values	BarCode is flagged enum with the types of barcodes, for example, **Code39**, **PDF417**, etc.
UseChecksum	Gets or sets a value that indicates whether or not barcode detection should use checksums.	bool	
EnableExactLength	Gets or sets a value that enables the length of barcodes detection.	bool	
ExactLength	Gets or sets a value that indicates the length of barcodes to be detected. Only barcodes having this length will be in the filter's output.	integer	Default value is 0 which indicated the barcode length must not be checked. Values larger than 0 indicate that only barcodes with this length are detected; all others are skipped.
MinHeight	Gets or sets the minimum height of barcode.	double	0 to 1000
Region	Specifies the region of interest on the image.	System.Windows.Rect	

2.6.3.1.2 DetectPatchcode

A binary filter enabling detection and decoding patchcodes on images.

Note: Test the filter on images similar to those used in production mode.

In some cases you may need to perform additional image processing before applying the filter. For example, use the following filters:

- **Deskew**: to straighten skewed images.
- **RemoveBackground**: to clean up dirty images.
- **RemoveLines**: repair the lines in the barcode itself.
- **SmoothEdges**: to smooth the lines in the barcode itself.

Table 2-14: DetectPatchcodes Filter Options

Property name (non-translated programmatic name)	Description	Type	Values
Direction	Gets or sets the direction that will be used to detect patch code.	string	• **Vertical**: Detects vertical patchcodes • **Horizontal**: Detects horizontal patchcodes • **Both**: Detects both horizontal and vertical types of patchcodes
MinBarWidth	Gets or sets the maximum width (or slice) of a patch code.	double	0 to 100
MaxBarWidth	Gets or sets the minimum width (or slice) of a patch code.	double	0 to 100
Region	Specifies the region of interest on the image.	System.Windows.Rect	

2.6.3.1.3 DetectBlankPage

A filter enabling detection of blank pages using *Preset* and *BlackAreaRatio* methods and specifying margins to exclude from the image when determining if the image is blank.

You can specify the precise number of black pixels to consider the page as blank. Ratio is the amount of black pixels divided by the amount of all region pixels.

Table 2-15: DetectBlankPage Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets method to be used for blank page detection.	string	• **BlackAreaRatio**: detection using *`BlackAreaRatio`* property • **Preset**: detection using *`Preset`* property
BlackAreaRatio	Gets or sets the amount of black pixels that allows determines if the page is blank.	float	0 to 1
DetectionPreset	Gets or sets one of the predefined presets which indicate that the image contains noise or data.	string	• **TwoLinesOK**: Blank pages contain two lines of data. • **OneLineOK**: Blank pages contain a single line of data. • **VeryDirtyWhite**: Blank pages contain a lot of noise. • **DirtyWhite**: Blank pages contain some noise. • **PristineWhite**: Blank pages don't contain noise.
LeftMargin	Gets or sets the left border excluded from the detection area.	double	0 to 10

Property name (non-translated programmatic name)	Description	Type	Values
RightMargin	Gets or sets the right border excluded from the detection area.	double	0 to 10
TopMargin	Gets or sets the top border excluded from the detection area.	double	0 to 10
BottomMargin	Gets or sets the bottom border excluded from the detection area.	double	0 to 10

2.6.3.1.4 DetectColorMarks

A filter enabling detection of color marks on scanned documents.

Table 2-16: DetectColorMarks Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets color code detection mode.	string	• **Color**: Detect by color. • **Saturation**: Detect by saturation.
ColorSensitivity	Gets or sets filter sensitivity for mark color. Works with Mode = Color	int	0 to 100
MarkColor	Gets or sets color of the mark to be detected. Works with Mode = Color	Media.Color	
MarkSaturation	Gets or sets the minimal mark saturation. Works with Mode = Saturation	int	0 to 255
MarkSize	Gets or sets the minimum mark size.	double	0 to 50
MergingDistance	Gets or sets maximum distance between regions.	double	0 to 50

Property name (non-translated programmatic name)	Description	Type	Values
Region	Specifies the region of interest on the image.	System.Windows.Rect	

2.6.3.1.5 DetectColorfulness

A filter enabling detection of certain color in an image by determining percentage of the image that is “colorful”.

The filter can be used before applying binary or color filters as the **DetectColorfulness** filter lets you improve the image processing time. For example, if the image is not a 24-bit color image, the image pixel data is not read, and the filter returns 0 as a **Colorfulness** value. Otherwise, **Colorfulness** is the percentage of samples.

Table 2-17: DetectColorfulness Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
SampleSize	Gets or sets the sample size.	int	1 to 7
Colorfulness	Gets or sets the amount of color a sample must contain to be identified as colorful.	int	0 to 256

2.6.3.2 Removal Filters

A set of filters that allow you to remove different distractions and redundant elements of the image.

2.6.3.2.1 RemoveBackground

A filter enabling detection of the background color of a scanned document, to drop it out or flatten while capturing all color information in the foreground.

Table 2-18: RemoveBackground Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets the action to be taken on the background.	string	• **Dropout**: replaces background with specific color • **Smooth**: removes scanning and paper non-uniformity • **Patterned**: removes background (for example, halftone or dither pattern) **Note:** If you select the **Pattern** mode, the Image Processor module will process color images as binary (black and white) images.
NewBackgroundColor	Gets or sets the color used in dropout mode.	Media.Color	
Sensitivity	Gets or sets how“aggressive”the algorithm is in determining if a pixel is a background pixel. This property affects the total amount of background pixels detected.	int	-20 to 20

2.6.3.2.2 RemoveBlackBars

A filter enabling removal of a black area around an image when using overscan option for scanning images. It reduces the physical size of the scanned image and the image file size, removing the black border (produced by scanners). For better results, use the **Deskew** filter and set the black **FillColor** before applying the filter.

Table 2-19: RemoveBlackBars Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets the operation mode of the filter.	string	• **RemoveBlackOverscan**: reduces image physical size by eliminating the black border generated by scanners with black backgrounds. • **ClearBlackOverscan**: inverts the overscan area of an image and image dimensions stay unchanged.
MaxProcessingLeft	Gets or sets the limit for the zone from the left side.	double	0 to 100
MaxProcessingRight	Gets or sets the limit for the zone from the right side.	double	0 to 100
MaxProcessingTop	Gets or sets the limit for the zone from the top.	double	0 to 100
MaxProcessingBottom	Gets or sets the limit for the zone from the bottom.	double	0 to 100

2.6.3.2.3 RemoveHoles

A binary filter enabling removal of binder holes around the image edges. The filter uses the following algorithm: the value is multiplied by page height to specify area to search for binder holes. For example, if you set the value to 0.125 when processing 11-inch image, filter will look for binder holes of 1.375 inches (11 inches x 0.125) in a specific part of the image (top, bottom, left, or right).

Table 2-20: RemoveHoles Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Bottom	Gets or sets the location to search for binder holes near the bottom edge of the image.	double	This value is multiplied by the height of the page to specify the area to search for binder hole objects. For example, when processing an 11-inch high image, a value of 0.125 would attempt to find objects in a horizontal band from the bottom edge of the image to a point 11 inches x 0.125, or 1.375 inches, from the bottom edge of the image. A fractional decimal value between 0 and 1
Left	Gets or sets the location to search for binder holes near the left edge of the image.	double	
Right	Gets or sets the location to search for binder holes near the right edge of the image.	double	
Top	Gets or sets the location to search for binder holes near the top edge of the image.	double	

Property name (non-translated programmatic name)	Description	Type	Values
Region	Specifies the region of interest on the image.	System.Windows.Rect	

2.6.3.2.4 RemoveLines

A binary filter enabling removal or reconstruction of lines on a form-based image. Filter allows you to reduce image file size and improve *OCR*.

Table 2-21: RemoveLines Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets the type of line correction to be performed on the image.	string	• **ReconstructForm**: removes lines, redraws straight lines, and reconnects lines that were previously connected. Commonly used for tables and forms. • **ReconstructLines**: removes lines, repairs overlapped graphics and text, and redraws straight lines in place of removed lines. • **RemoveLines**: removes lines.
HorzEnable	Gets or sets a value that specifies whether or not horizontal lines should be detected.	bool	

Property name (non-translated programmatic name)	Description	Type	Values
HorzMaxGap	Gets or sets the maximum white space allowed between two horizontal line-like objects for them to be considered a single line.	double	0 to 100
HorzMinLength	Gets or sets the minimum detectable length of horizontal lines.	double	0.003 to 100
HorzStraightLine	Gets or sets a value that specifies whether or not to use the straight-line processing algorithm on horizontal lines.	bool	
HorzCurvature	Gets or sets the maximum deviation from a straight line allowed for a horizontal line-like object to be considered a line.	int	0 to 100
VertCurvature	Gets or sets the maximum deviation from a straight line allowed for a vertical line-like object to be considered a line.	int	0 to 100
VertEnable	Gets or sets a value that specifies whether or not vertical lines should be detected.	bool	
VertMaxGap	Gets or sets the maximum white space allowed between two vertical line-like objects for them to be considered a single line.	double	0 to 100

Property name (non-translated programmatic name)	Description	Type	Values
VertMinLength	Gets or sets the minimum detectable length of vertical lines.	double	0.003 to 100
VertStraightLine	Gets or sets a value that specifies whether or not to use the straight-line processing algorithm on vertical lines.	bool	
Region	Specifies the region of interest on the image.	System.Windows.Rect	

2.6.3.3 Color Adjustment Filters

A set of filters that can be used for establishing further sequence of filters (for example, color content detection or color auto detection) and scanner configuration (for example, in scanner control sheets).

2.6.3.3.1 AdjustOverallColor

A filter enabling image color adjustment.

Table 2-22: AdjustOverallColor Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Algorithm	Gets or sets property to set the *`Autoadjustment`* algorithm.	string	• **Nonlinear** uses *`HistogramEqualization`* algorithm • **Linear** uses *`ContrastStretching`* algorithm
Mode	Gets or sets property to set the mode of a filter, automatic or curves	bool	

Property name (non-translated programmatic name)	Description	Type	Values
ApplyMode	Gets or sets the apply mode for algorithm, by channel or by luminosity.	string	• **ByLuminosity**: treat by image luminosity. • **ByChannel**: treat each channel separately.
LinearMode	Gets or sets a linear algorithm mode. Is used only when algorithm is set to Linear.	string	• **Cutoff Fraction**: first occurrence of value bigger than histogram preset Used if the Linear algorithm is selected. • **Percentiles**: 5% to 95% percentiles used of the histogram.
Polynom	Gets and sets the polynom for a conversion for separate channel.	double[][]	Polynom represents Lagranzh polynom points for advanced color adjustments. These are set of coefficients for each color channel: RGB, R, G, B.

2.6.3.3.2 ConvertSpecificColor

A filter enabling conversion of particular colors into other colors.

Table 2-23: ConvertSpecificColor Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
OldColor	Gets or sets the color to find.	Media.Color	
NewColor	Gets or sets the color to substitute with.	Media.Color	
Magnitude	The value that indicates the radius of the color sphere.	int	1 to 255

2.6.3.3.3 ConvertToBlackWhite

A filter enabling conversion of 24-bit color images to a binary image.

Note: Filter uses a specific threshold for changing the pixels in a color image: Images that are darker than the specified brightness and contrast, are converted to black; all pixels lighter than the threshold are converted to white.

Table 2-24: ConvertToBlackWhite Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Brightness	Gets or sets the desired brightness of an image.	int	1 to 100
Contrast	Gets or sets the desired contrast of images.	int	1 to 100
Dither	Gets or sets indicator whether or not to dither the image.	bool	
Mode	Gets or sets a trade-off between quality and performance.	string	• **BinaryFast**: fastest mode but less accurate • **BinaryBalanced**: optimal speed and accuracy • **BinaryAccurate**: maximum accuracy with lower speed • **Grayscale**: produces grayscale output image instead of binary image

2.6.3.3.4 ConvertToBlackWhiteAdvanced

A filter enabling detection of the images colorfulness. If the input image is less colorful than a specified value, it is thresholded and output as a binary image.

The filter operates the following way:

1. Considers image background.
2. Counts pixels that are white and nearly white and pixels that are black or nearly black excluding these from count of pixels that are colorful.
3. Counts the colorfulness of the image according to the Detect Type property value.
4. Resulting value is output in PixIpResult object and compared to value specified in Threshold property. If the colorfulness value is less than Threshold, then the image is thresholded, and the filter outputs a thresholded (binary) version of the original image.

Table 2-25: ConvertToBlackWhiteAdvanced Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Conversion mode:	string	• **None**: no specific mode. • **Fast**: the fastest mode, but less accurate. • **Balanced**: balanced speed and accuracy. • **Accurate**: maximum accuracy with lower speed.
Brightness	Gets or sets the desired brightness of thresholded images.	int	1 to 100
Contrast	Gets or sets the desired contrast of thresholded images.	int	1 to 100
DetectType	Gets or sets the type of color detection.	string	• **Ratio**: Ratio of color and black pixels. • **Amount**: Amount of color pixel in image.

Property name (non-translated programmatic name)	Description	Type	Values
Dither	Gets or sets whether or not to dither thresholded images.	bool	
IgnorePaperColor	Gets or sets whether or not to detect and remove a colored background before performing automatic color detection.	bool	
Threshold	Gets or sets the amount of color a sample must contain to be identified as "colorful".	int	0 to 100

2.6.3.3.5 InvertBlackWhite

A filter enabling inversion of a binary image to its negative equivalent actually modifying the image data: white pixels are changed to black and vice versa.

Table 2-26: InvertBlackWhite Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Region	A specific region of interest on the image.	System.Windows.Rect	

2.6.3.4 Image Quality Filters

A set of filters used to enhance the quality of the image.

2.6.3.4.1 AdjustLighting

A filter enabling adjusting brightness and contrast properties for the image.

Table 2-27: AdjustLighting Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Brightness	Gets or sets the desired brightness adjustment of an image.	int	-50 to 50
Contrast	Gets or sets the desired contrast adjustment of an image.	int	-50 to 50

2.6.3.4.2 AdjustThickness

A binary filter enabling adjustment of black objects.

Table 2-28: Adjust Thickness Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Level	Sets the level of adjustment.	int	• Value 0: reduces the size of text and image graphics in your batch images to one-pixel thick skeletons of the original objects. • Values 1 to 10 to reduce the size of text and image graphics in your images. • Values 12 to 21 to increase the size of text and image graphics in your images.

Property name (non-translated programmatic name)	Description	Type	Values
Direction	Gets or sets the direction to adjust thickness.	Comma separated string of values	• **None** • **Diagonal** • **Horizontal** • **Vertical**
Region	Specific image region to adjust thickness.	System.Windows.Rect	

2.6.3.4.3 RemoveSpecks

A binary filter enabling you to set maximum area percentage to consider the object as a speck or noise. A filter requires setting maximum width and height to consider the noise as a speck.

Table 2-29: RemoveSpecks Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
MaxAreaPercent	Gets or sets the maximum percentage of the area defined by **MaxHeight** and **MaxWidth** that an artifact can occupy to be removed as a noise.	int	0 to 100
MaxHeight	Gets or sets the maximum height of a speck below which it will be removed.	double	0 to 100
MaxWidth	Gets or sets the maximum width of a speck below which it will be removed.	double	0 to 100
MinDistance	Gets or sets the minimum distance separating a speck from other artifacts above which it will be removed.	double	0 to 100
Region	Specific image region to remove specks.	System.Windows.Rect	

2.6.3.4.4 SmoothEdges

A binary filter enabling removal of bumps and spurs appearing on image text or graphics. The filter algorithm is the following: it looks for any pixel surrounded by five or six pixels of the opposite color and inverts the center pixel based on the filter configuration.

Table 2-30: SmoothEdges Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets the binary smoothing mode flag that control the smoothing operation to be performed.	string	• **None**: does not remove any noise. • **CornerWhite**: removes white noise pixels from the corners of objects. • **CornerBlack**: removes black noise pixels from the corners of objects. • **TrimFirst**: removes black noise pixels before removing white ones. If this flag is not set, then white noise pixels are removed before black ones.
Region	The region to apply smoothing to.	System.Windows.Rect	

2.6.3.5 Page Correction Filters

A set of filters that allow you to modify and correct the page layout and its properties.

2.6.3.5.1 Crop

A filter enabling setting the image margin and removing white spaces around the edge of the image.

Table 2-31: Crop Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets the mode the filter uses to set margin values.	string	• **CropMargins** • **DetectOnly** • **Fixed** • **Auto**
Top	Gets or sets the desired top margin after cropping.	double	-50 to 50
Left	Gets or sets the desired left margin after cropping. To reduce the margins from the image, the values should be negative.	double	-50 to 50
Right	Gets or sets the desired right margin after cropping. To reduce the margins from the image, the values should be negative.	double	-50 to 50
Bottom	Gets or sets the desired bottom margin after cropping. To reduce the margins from the image, the values should be negative.	double	-50 to 50
FixedHeight	Gets or sets the desired image height in inches.	double	Positive value
FixedWidth	Gets or sets the desired image width in inches.	double	Positive value

Property name (non-translated programmatic name)	Description	Type	Values
ForceSymmetry	Gets or sets whether or not the top/bottom and left/right margins should be made the same.	bool	• **Centre** • **Right** • **Left**
HorzAlignment	Gets or sets the alignment of the image on the horizontal axis.	string	• **Centre** • **Bottom** • **Top**
VertAlignment	Gets or sets the alignment of the image on the Vertical axis.	string	

2.6.3.5.2 Deskew

A filter enabling setting of the skew angle. This filter can be used to straighten a binary or color images that show a slant from their correct orientation. The filter returns the angle that the image was skewed. Skewing can occur if the original document was skewed when it was fed into the scanner, fax machine, or photocopier. The Deskew filter examines the image and determines the skew angle. The skew angle is measured between the actual edge of the image data and the horizontal or vertical axis. The image data is then rotated to correct the skew angle. Deskewing an image makes the image contents more legible and can improve OCR results. To achieve the best image quality after image processing and to find the optimal balance between processing speed and accuracy, you should test this filter with sample images that are similar to the real images you expect to process, and then fine tune the parameters for each filter to find the values most suitable for your document type.

Table 2-32: Deskew Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets the operation mode of the Deskew filter.	string	• **DetectOnly**: Detect angle • **FixedAngle**: Rotate by a fixed angle. • **DetectAndDeskew**: Detect angle and deskew.

Property name (non-translated programmatic name)	Description	Type	Values
Direction	Gets or sets the direction of the skew angle measurement.	string	• **Both** • **Horizontal** • **Vertical**
Features	Gets or sets image type that contains lines of text or graphics.	string	• **Image** • **Text**
FillColor	Gets or sets the color for the area at the edge of the image to be deskewed.	Media.Color	
FixedAngle	Gets or sets an arbitrary value of a fixed angle to rotate images, rather than automatically detecting and correcting skew.	double	0 to 360
Quality	Gets or sets the tradeoff between quality and performance.	string	• **Good** • **Fast**
RemoveShear	Gets or sets whether to remove shearing or rotational skew from the page. Works for binary images only.	bool	

2.6.3.5.3 Rotate

A filter enabling adjustment of binary and color images orientation and mirroring by flipping the page across the vertical axis. The adjustment increment is 90 degrees. The Rotate filter adjusts binary and color images orientation in 90-degree increments. It can rotate the image 90-degrees clockwise and counterclockwise. It can also rotate it 180-degrees as well as perform mirroring, which flips the page across the vertical axis so that it appears to be the mirror image of the original. It can also constrain orientation to specified one (Landscape or Portrait).

Table 2-33: Rotate Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Gets or sets the desired image rotation/orientation.	string	• **LandscapeOnly**: constrain landscape orientation • **PortraitOnly**: constrain portrait orientation • **Rotate180**: rotate 180 degrees • **CounterClockwise**: rotate counterclockwise • **Clockwise**: rotate 90 • **None**: no rotation is performed
Mirror	Gets or sets whether or not to mirror the image.	bool	

2.6.3.5.4 Scale

A filter enabling scaling an image and changing its dimensions (DPI). You can use the filter for the following purposes:

- Leveraging image resolution
- Modifying image dimensions
- Preparing for further archiving

Table 2-34: Scale Filter Properties

Property name (non-translated programmatic name)	Description	Type	Values
Mode	Scaling mode selection:	string	• **ScaleToSize**: Scales image to specified size with the same resolution. • **ScaleToResolution**: Scales image to specified resolution with the same size. • **ScaleWithCoefficient**: Scales image with specified coefficient. Resolution remains the same. • **ResolutionAlignment**: Changes resolution in image properties so that horizontal and vertical resolutions are equal. In this mode, maximum one is selected for a new resolution. Bitmap itself remains as is. • **ChangeResolution**: Changes resolution in the image properties; bitmap itself remains as is resulting in updated resolution and size.
Height	New height available with **Scale To Size** mode selected.	double	
Width	New width available with **Scale To Size** mode selected.	double	

Property name (non-translated programmatic name)	Description	Type	Values
PageResolution	New resolution available with **Scale To Resolution** mode selected.	int	
ScaleFactor	Specifying scale factor for Scale with **Coefficient** mode.	float	1 to 10
Smoothing	Defines whether to perform scaling with smoothing.	bool	
RoundWidth	Defines whether page width (in pixels) must be rounded to multiple. For example, if *`RoundWidthTo`* is 16 then resulting image width (in pixels) will be a multiple of 16.	bool	
RoundWidthTo	Defines multiple for *`RoundWidth`*.	int	

2.7 Custom Filter Scripts

This section describes how to create, deploy, and debug custom filters.

2.7.1 Creating a Custom Filter

Customers may create their own custom filters and use them in **Image Processing** designer. A sample custom filter script is located at: `<<drive>>:\Program Files\InputAccel\Client\src\Sample Capture System\ScriptSource\Image Filter`.

To create a custom filter:

1. Create a Visual Studio project for a C# or Visual Basic Class Library.
2. Edit project properties:

    a. Set the **Assembly name** to any value that matches the pattern `Custom.ImageFilter.<your name>.dll`, such as `Custom.ImageFilter.MyCustomFilter.dll`.

    b. Set **TargetFramework** to **4.8**.

    c. Ensure **Output Type** is **Class Library**.

d. Set **Build output path** to `<Path-to-client-install>\Client\binnt`. The custom filter DLL must be in the same directory as Intelligent Capture Designer and Image Processor.

3. Add a reference to `<path-to-client-install>\Client\binnt\Emc.InputAccel.CaptureClient.dll`. This DLL resolves references to all context interface objects and classes needed by the script.

4. Implement the abstract class `ImageFilterFactory`.

5. For each custom filter, a separate class must be implemented. This classes must inherit the `ImageFilter`abstract class.

6. If the custom filter contains some editable properties, they must be added as static read-only fields. These properties must be created with the `CustomFilterProperty` constructor. Available types of properties are: `integer`, `double`, `color`, `rectangle`, and `choice`. For example, a `choice` property for selecting a rotation angle might look like:

```
/// <summary>
/// Defines angle of rotation
/// </summary>
public static readonly CustomFilterProperty Angle = new CustomFilterProperty(
     "Rotate to",
     "Rotates an image for 90,180 or 270 degrees",
     "choice",
     "None,90,180,270");
```

After deploying custom filters, they are available in the **Custom Filters** category in the **Image Processor** area of Intelligent Capture Designer.

2.7.2 Deploying Custom Filters

To deploy custom filters:

1. Create and compile the custom filter script in Visual Studio.

2. Copy the DLL and data files to `<path-to-client-install>\Client\binnt` folder of the Intelligent Capture Designer machine and the Image Processor client machine.

3. Restart Image Processor client modules.

2.7.3 Debugging Custom Filters

To debug custom filters:

1. Create and compile the custom filter script in Visual Studio.
2. Copy the DLL and data files to `<path-to-client-install>\Client\binnt`.
3. Create an Image Processing profile that uses the custom filter.
4. Configure Visual Studio to launch Intelligent Capture Designer when debugging the DLL and set breakpoints as needed in the source.
5. Select **Start Debugging** in Visual Studio and add open the Image Processor profile to cause the filter to execute.

2.8 Batch Creation Scripts

Batch creation scripts provide control over the batch creation process in Standard Import (Email Import and File System types only).

For more information about profile script class and DLL structure, see "Architecture" on page 27.

2.8.1 Overview

Batch creation scripts provide control over the batch creation process in Standard Import (Email Import and File System types only) by changing the batch's polling schedule, contents, and structure.

You create a class with event handlers to process batch creation events. The class must extend Emc.InputAccel.UimScript.UimScriptBatchCreate, which is defined in `Emc.InputAccel.CaptureClient.dll`.

For more information about the script class and DLL structure, see "Overview" on page 27.

If a Standard Import profile (Email Import and File System types only) is configured for scripting, then when a task is received, the runtime looks for the specified script class in the DLLs, creates a new instance, and then invokes the appropriate events, either CanStartCycle or ProcessList. If the class is not found, then the profile is not used and an error is reported.

Note: Script code must not expect thread affinity; that is, there is no guarantee that the same thread is used for all batch creation tasks.

2.8.2 Context Objects Passed to Event Methods

When an event method is invoked, it is passed a runtime context object. This context object is used to access runtime data. All context objects are passed as interfaces and these interfaces are defined in `Emc.InputAccel.CaptureClient.dll` with the `Emc.InputAccel.UimScript` namespace.

Batch creation scripts receive the following types of context objects:

Context Object	Description
`IUimScriptBatchCreateServiceContext`	Provides information about the profile and its run state.
`IUimScriptBatchCreateListContext`	Provides information about and control over the batch that is being created.

2.8.3 Batch Creation Class Template Example

Note: This sample represents an **Email Import**type of **Standard Import Profile** script; the only change for a **File System**type of **Standard Import Profile** script would be the name of the class.

```
namespace Custom.InputAccel.UimBatchScript
{
    using Emc.InputAccel.UimScript;

    /// <summary>
    /// Custom script class for email import profile <Mail Profile Name as
    /// defined in Intelligent Capture Designer>.
    /// </summary>
    public class ScriptMail<Mail Profile Name as defined in Intelligent Capture
Designer> : UimScriptBatchCreate
    {
        /// <summary>
        /// Default constructor.
        /// </summary>
        public Script<Mail Profile Name as defined in Intelligent Capture Designer>()
            : base()
        {
        }

        /// <summary>
        /// Checks whether a batch creation cycle should be started.
        /// </summary>
        /// <param name="context">Information about the running profile</param>
        /// <returns>True to start a batch creation cycle</returns>
        public bool CanStartCycle(IUimScriptBatchCreateServiceContext context)
        {
        }

        /// <summary>
        /// Checks whether a batch creation cycle should be started.
        /// </summary>
        /// <param name="context">Information about the batch</param>
        /// <param name="items">The items that can be used in the batch</param>
        public void ProcessList(IUimScriptBatchCreateListContext context,
                                IList<IImportNodeItem> items)
        {
        }
```

```
    }
}
```

2.8.4 Batch Creation Script Reference

2.8.4.1 CanStartCycle

Syntax

```
C#: public abstract bool CanStartCycle(IUimScriptBatchCreateServiceContext context);
```

```
VB.NET: Public MustOverride Function CanStartCycle(context As
IUimScriptBatchCreateServiceContext) As Boolean
```

Description

This event is executed when Standard Import is checking whether the content should be collected and a batch created. This event is used only if the polling expression in the profile configuration is empty.

Use Cases

This event can be used to implement more complex logic than the polling rule easily supports by itself. Examples of this kind of complex logic are:

- Time-of-day variation: for instance, creating batches more quickly near the end of the day to ensure that content is captured more quickly.
- State-based variation: for instance, creating batches more slowly after a batch has just been created.

2.8.4.2 ProcessList

Syntax

```
C#: public abstract void ProcessList(IUiScriptBatchCreateListContext context,
IList<IImportNodeItem> items);
```

```
VB.NET: Public MustOverride Sub ProcessList(context As
IUiScriptBatchCreateListContext, items As IList(Of IImportNodeItem))
```

Description

This event is executed when Standard Import is about to create a batch from the supplied items or a subset of them.

Use Cases

This event can be used to control the structure and IA values in the new batch. For example:

- Dividing documents into folders based on their names or contents.
- Duplicating header sheets to multiple folders or documents in the batch.

- Creating entirely new images.
- Adding data to the batch based on the file names, contents, and create date for the batch.
- Deferring files from this batch to be reconsidered in the next batch.

2.9 Batch Filtering Scripts

Batch filtering is an optional feature that can be implemented for operator modules such as Completion and Identification. When this option is implemented and the module is launched from the command line with the *-customtaskbatchlistfilter* parameter, the operator views the filtered list of batches available for processing.

This parameter can be used with the *-autostart* command line parameter (to run the **Run All Batches** mode) or when the operator selects the **Run All Batches** mode. In this case, only tasks from the filtered list of batches (instead of from all batches) are returned from the Intelligent Capture Servers.

To implement batch filtering, you need to include the `ModuleBatchListView` method in the `ScriptMain` class.

The `ModuleBatchListView` method is called before the batches with tasks are returned to the module and displayed to the operator in the batch list. The batch list can be filtered and resorted by removing items and changing their order in the list. Scripting must not change the content of any rows in the list. The content must be treated as read-only.

Example

```
using System;
using System.Collections.Generic;
using System.Windows;
using Emc.InputAccel.CaptureClient;
using Emc.InputAccel.UimScript;

namespace Custom.InputAccel.UimScript
{

    public class ScriptMain : UimScriptMainCore
    {
        public ScriptMain()
            : base()
        {}


        protected override void ModuleBatchListView(string moduleType, List<ITableRow>
tableRowList, string loginName,
string[] departments)
        {
            MessageBox.Show("moduleType: " + moduleType + Environment.NewLine +
                            "loginName: " + loginName  + Environment.NewLine);

            string DepartmentsToShow = "";
            foreach (var entry in departments)
            {
                DepartmentsToShow += entry + ",";
            }
```

```
            MessageBox.Show("Departments: " + DepartmentsToShow);

            for (int index = 0; index < tableRowList.Count; index++)
            {
                /*Row columns are
                  0 Batch Name
                  1 Batch ID
                  2 Tasks in the Batch
                  3 Batch Creation date
                  4 Batch Priority
                  5 Batch Description for English locale.
                */
                var answer =  MessageBox.Show(
                   "Batch Name: " + tableRowList[index].AsString(0) +
Environment.NewLine +
                   "ID: " + tableRowList[index].AsString(1) + Environment.NewLine +
                   "Tasks: " + tableRowList[index].AsString(2) + Environment.NewLine +
                   "Created: " + tableRowList[index].AsString(3) + Environment.NewLine +
                   "Priority: " + tableRowList[index].AsString(4) + Environment.NewLine +
                   "Batch Description: " + tableRowList[index].AsString(5) +
Environment.NewLine
                    , "Remove this batch from List?", MessageBoxButton.YesNo
                   );
               if (answer == MessageBoxResult.Yes)
                {
                  // Do not display the batch
                  tableRowList.RemoveAt(index); index--;
                }
            }
        }

        /// Called when the script subsystem is first loaded before any other events are
fired.
        /// The implementing class can override this method to provide additional
initialization.
        /// An overridden implementation must always call base class method before doing
anything else.

        public override void ScriptLoad()
        {
            base.ScriptLoad();
            // Add your initialization here
        }

        /// Called when the script subsystem is unloaded.
        /// The implementing class can override this implementation to provide
        /// additional unload-related clean up. An overridden implementation must always
        /// call the base class after doing everything else.

        public override void ScriptUnload()
        {
            // Add your unload code here
            base.ScriptUnload();
        }
    }

}
```

2.9.1 Batch Filtering Script Reference

2.9.1.1 ModuleBatchListView

Syntax

```
C#: protected virtual void ModuleBatchListView(string moduleType,
List<ITableRow> tableRowList, string loginName, string[] departments)
```

```
VB.NET: Protected Overridable Sub ModuleBatchListView(moduleType As String, tableRowList
As List(Of ITableRow), loginName As String, departments As String())
```

Parameters

- *`moduleType`*: The client module type. Example: CPDSKTOP
- *`tableRowList`*: The list containing batch rows. The row columns for the English locale are: *`Batch Name`*, *`ID`*, *`Tasks`*, *`Created`*, *`Priority`*, and *`Batch Description`*. The order of the columns is fixed. To access data in the columns, use the 0 based column indexing.

  For example, index value "1" references *`BatchID`*, index value "5" references *`Batch Description`*. The batch description is not shown in the user interface but is provided here for programmatic access.

  Note: Using the column names for data access is not recommended because the column names are localized to the calling module's user interface language.
- *`loginName`*: The login name of the module user.
- *`departments`*: The departments in which the module is working.

Chapter 3

Task Scripting

Task scripting is used for task and batch node manipulation.

Note: For more information about adding custom behavior to a CaptureFlow process using the .NET Code module, see *OpenText Intelligent Capture - .NET Code Module Guide (ECPCORE-CNT)*.

3.1 CaptureFlow Scripting

CaptureFlow scripting is done using the `Emc.InputAccel.CaptureFlow` namespace. The complete programming reference is available in the Document and Task Scripting section.

3.1.1 Unsupported CaptureFlow Scripting Features

You can use most of the standard functions provided with C#/VB.NET in your *XPP*, with the following exceptions:

Table 3-1: Unsupported CaptureFlow Scripting Features

Element	Description
Screen input and output functions	Screen input and output functions such as `MsgBox` or `InputBox` is not supported. If you include these in your process, then the resulting user interface only displays on the Intelligent Capture Server console, not on the client machine. If the option requires user input, production is halted and the operator must have direct access to the Intelligent Capture Server console to resume processing.

Element	Description
Network and *CPU*-intensive .NET operations	All CaptureFlow scripting execution is single-threaded within an Intelligent Capture Server instance and runs inside the Intelligent Capture Server process. Thus, you must avoid possible long-running and high-load operations, such as: • *ODBC* calls, access to external back-end systems, such as Documentum, SAP, SQL Server, and others, • Calls to an external system using a Web service, • Local and network file access and other network operations, • Calls to any third-party libraries or external processes, • Long-running computations, complex data analysis, and other *CPU* and memory-intensive operations. Instead, put this type of code in a script within a client module or use a dedicated .NET Code module.
Global and Static variables	Using `Global` and `Static` variables in an *XPP* is not supported. Their life span is indeterminate and they generally cause unpredictable effects. These variables sometimes retain their value between events, so they can appear persistent. However, whenever the CaptureFlow project is unloaded, their values are lost. The *XPP* designer cannot know when the CaptureFlow project is unloaded and this may change in future versions of the Intelligent Capture Server. **Note:** A variable is "persistent" if it retains its value when the Intelligent Capture Server syncs, the batch in which it is used is unloaded, or if the Intelligent Capture Server is shut down and restarted. IA values are persistent, but variables declared within C#/VB.NET are not. This is true even if you use the Global and Static keywords provided by the C#/VB.NET language.
External code assemblies	Use of the code assemblies that are not installed with the product is not supported.
Direct *DLL* calls	Use of direct *DLL* calls is not supported.

Element	Description
Setting trigger variables	Setting trigger variables is not supported.
Adding a CS/VB file to the *XPP*.	Adding CS/VB files to the CaptureFlow project is not supported. A file in the project is generated when the *XPP* designer adds scripting to a particular step on the CaptureFlow diagram. However, you can add auxiliary code to the existing CS/VB files.

Chapter 4

Intelligent Capture REST Services

4.1 Overview

Intelligent Capture Real Time Services is a product offering based on Intelligent Capture REST Services, which are a set of RESTful web service interfaces that custom client applications can use to call the services of the Intelligent Capture Server or the Module Server. An example of an Intelligent Capture REST Services client is Intelligent Capture Web Client.

You use the Intelligent Capture REST Services in your application to perform a batch request in a CaptureFlow or an Ad Hoc Service request for Module Server services as follows:

- In a batch request, your application sends documents and data to the Intelligent Capture REST Service Web application, which creates an Intelligent Capture batch, adds the documents and data to the batch, and then sends the batch to the Intelligent Capture Server, which executes the specified CaptureFlow. You can also write a custom Intelligent Capture REST Service Web application authentication plug-in that authenticates and maps the Intelligent Capture REST Service Web application's callers to the appropriate Intelligent Capture user roles.
- In an Ad Hoc Service request, your application makes a request to the Intelligent Capture REST Service Web application for Module Server services, such as classifying and extracting pages or reading barcodes.

Note: Hereafter, the Intelligent Capture REST Service Web application's authentication plug-in is called the authentication plug-in.

4.2 Architecture

The Intelligent Capture REST Services architecture consists of the following tiers:

- Client tier
  - Intelligent Capture Web Client
  - Your custom client applications; for example, a mobile application that is installed onto mobile phones or tablets (Android or iOS).
- Web tier
  - The Intelligent Capture REST Service Web application and, optionally, your custom authentication plug-in, which reside on Web servers (including Web farms).

Client-side scripting is supported. See *Intelligent Capture Web Client Scripting Development Guide*.

- A shared file system for temporary image capture storage and the Intelligent Capture REST Service Web application and Module Server's configuration files.

- Application tier
  - The Intelligent Capture system.

    Intelligent Capture server-side scripting is supported. See "Intelligent Capture Web Client Server-side Document Type Scripting" on page 127.
  - The Module Server.

    The Module Server provides the following services:

Module Service	Ad Hoc Service
Classification and Extraction	Classify Service Classify Extract Document Service Classify Extract Page Service Extract Document Service Extract Page Service
Full Page OCR	Full Page OCR Service
Image Conversion	Convert Images Service
Image Processing	Process Image Service Read BarCodes Service
Data Population and Validation	UimData Service

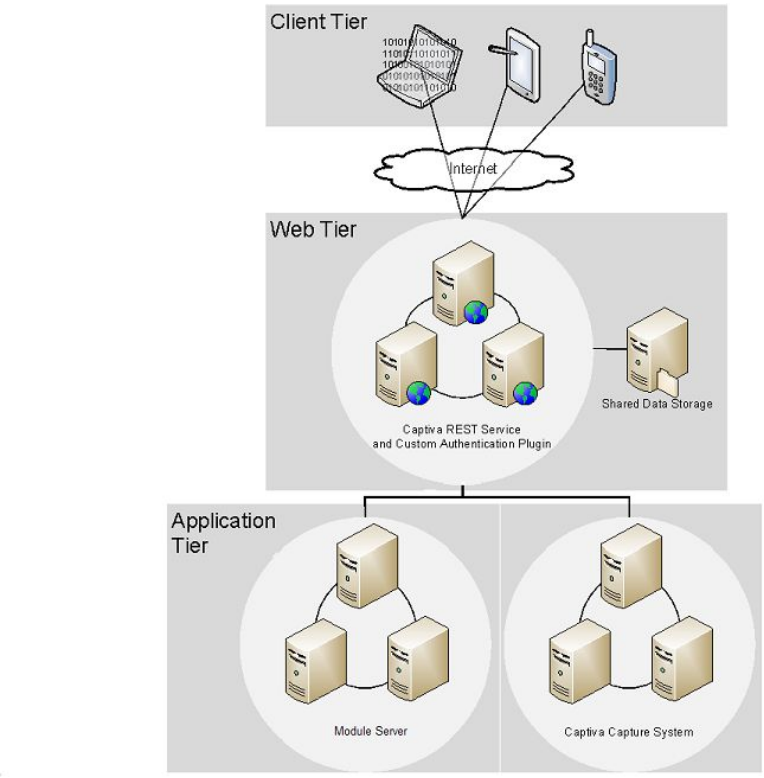


Figure 4-1: Intelligent Capture REST Services Architecture

4.3 Locations

Table 4-1:

Components	Contents	Location
Intelligent Capture REST Service Documentation and Samples	Intelligent Capture REST Services reference documentation (including *Intelligent Capture-Scripting Guide*)	`C:\Program Files\InputAccel (x86)\Documentation\pdf`
	Intelligent Capture REST Service client sample	`C:\Program Files (x86)\InputAccel\WebComponents\Samples` (default)
	Intelligent Capture REST Service authentication plug-in sample	`C:\Program Files (x86)\InputAccel\WebComponents\Samples` (default)

Components	Contents	Location
Intelligent Capture REST Service	• Intelligent Capture REST Service Web application • Intelligent Capture REST Service configuration tool	Installed on an IIS server as a Web application using the Intelligent Capture installer. See the *OpenText Intelligent Capture - Installation Guide (ECPCORE-IGD)*. **Note:** For specific system requirements, see the *Intelligent Capture Release Notes*.
Module Server	Module Server Windows service	Installed on a Windows machine using the Intelligent Capture installer. See the *OpenText Intelligent Capture - Installation Guide (ECPCORE-IGD)*. **Note:** For specific system requirements, see the *Intelligent Capture Release Notes*.

4.4 Procedure

In general, to send batches of captured documents to an Intelligent Capture Server or the Module Server, your application sends the following requests to the Intelligent Capture REST Service Web application in the following order:

Note: For the complete API specification and request and response syntax, see the *Intelligent Capture REST Services Development Guide*.

1. Retrieve the home document.
2. Log in.
3. Submit a batch or an Ad Hoc services request:

    - Batch

        a. (Optional) Retrieve the names of the CaptureFlows to be used for the creation of the batch.

        b. Create a batch.

        c. Add or remove captured documents and data to/from the specified batch.

        d. Submit the batch to the Intelligent Capture Server.

        You should make sure to manage the number of batches that reside on the Intelligent Capture REST Service. For example, it is a best practice to delete batches from the Intelligent Capture REST Service after they have been

submitted to the Intelligent Capture Server. The maximum number of batches that can reside on the Intelligent Capture REST Service is as follows:

- For batches that have already been submitted to the Intelligent Capture Server, the maximum is 10 per session. A batch is submitted to the Intelligent Capture Server when the REST API update batch call has the `dispatch` property value set to `S` (or the **Submit** button in the Intelligent Capture Web Client is clicked).

  If the maximum (or higher) is reached, then the number of submitted batches required to bring the number down to less than 10 are deleted. No error is generated.

- For batches that have not yet been submitted to the Intelligent Capture Server, the maximum is 4 per session.

  If the maximum is reached, then the following error is returned:

  ```
  Emc.InputAccel.Core.CoreRuntimeException: The session has reached maximum
  number of pending batches
  ```

- Ad Hoc services request

  a. (Optional) Add or remove captured documents and data to/from the specified Files resource.

  b. Submit the documents and data to the Module Server in one of the following ways:

  - Referencing the Files resource in the Ad Hoc Services call.
  - Adding the documents and data directly within the request body of the Ad Hoc Services call.

4. Log out.

4.5 Intelligent Capture REST Services Sample Application

The sample application has a user interface that enables you to exercise all of the Intelligent Capture REST Services functionality. Because it displays the JSON request and response for each request, it is an excellent way to learn how to implement those same requests in your custom client application.

Note: To see actual request and response behavior in the sample application, make sure that the Intelligent Capture REST Service Web application is running and that it is correctly configured to connect to an Intelligent Capture Server and the Module Server.

4.6 Intelligent Capture REST Service Web Application's Authentication Plug-in

To limit exposure to security risks, a custom authentication plug-in provides a level of indirection by mapping users to Intelligent Capture user roles. For example, your custom mobile application logs into your corporate authentication server and obtains a ticket. Your mobile application passes this ticket in the `ExtraAuthInfo` parameter in a call to the Intelligent Capture REST Service Web application. Your custom authentication plug-in processes this ticket by contacting your corporate authentication server, obtaining the user roles, mapping them to Intelligent Capture user roles, and completing the authentication process.

Note: A custom authentication plug-in could also simplify configuration and user management.

To create your custom authentication plug-in, you create a .NET DLL that includes a class that implements the `Emc.InputAccel.CaptureClient.AuthenticationPlugin` abstract class, including the `AuthenticateAndMapToInputAccelUserInfo` method that takes a user's credentials (as well as any extra authentication information) and returns one or more Intelligent Capture role names, which are to provide the security information needed for the service, and the departments with which the user is associated.

Note: For the minimum permissions of the Intelligent Capture roles to which to map the Intelligent Capture REST Service Web application's users, see the *Intelligent Capture Installation Guide*.

To use your custom authentication plug-in from the Intelligent Capture REST Service Web application, you specify its path in the Intelligent Capture REST Service Web application's **Sites** > **Intelligent Capture REST Service** > **ASP.NET** > **Application Settings** > **CaptivaAuthPlugin** setting (in the IIS Management Console).

If you are using the `Emc.InputAccel.CaptureClient.AuthenticationPlugin.CustomParameter` property, then you must set it in the **Sites** > **Intelligent Capture REST Service** > **ASP.NET** > **Application Settings** > **CaptivaAuthPlugin** after the custom authentication plug-in path as follows:

```
<custom_plugin_path>;<custom_param_value>
```

For the exact implementation details of the `Emc.InputAccel.CaptureClient.AuthenticationPlugin` class, see its reference documentation.

4.6.1 Sample

This sample application demonstrates a custom authentication plug-in that reads from a text file. The full source code is included.

Note: Although you can use any IDE, this sample is provided as a Visual Studio 2013 project.

4.7 Launching Intelligent Capture Web Client From Another Application

You can launch Intelligent Capture Web Client from another application using a URI as follows:

Notes

- Unless otherwise noted, these parameters can only be used with `Home.aspx`.
- For more information about the parameter values, see the *Intelligent Capture REST Services Development Guide*.
- Hereafter, Intelligent Capture Web Client is referred to as Web Client.

Table 4-2: Web Client URL Parameters

Parameter	Description
`appvalues=<name1>:<value1>,<name2>:<value2>,...`	Name-value pairs to push root-level values into the batch.
`autologout=[true|false]`	Set to `true` to automatically log the user out after submitting the current batch and immediately prior to displaying the batch name in the batch submission status panel. By default, the user is returned to the login page (`login.aspx`). To redirect the user to a specific URL, specify *`logoutret`*. **Note:** Line-of-business applications can take advantage of this parameter.
`batchid=<string>`	The ID of a specific batch to be opened for processing.
`cptvticket=<string>`	A valid `CPTV-TICKET` used for authentication when retrieving protected pages. If `CPTV-TICKET` is invalid, then the user is redirected to the login page.

Parameter	Description
`culture=<culturecode>`	Overrides the culture specified in the `Accept-Language` header. **Note:** Can be used with any page.
`eai=<string>`	A string to be passed to the *`extraAuthInfo`* parameter for the Intelligent Capture REST Service authentication plug-in.
`logoutret=<url>`	The URL to which to return after logging out of Web Client. The user is not logged out of the Intelligent Capture REST Service.
`profile=<profile-name>`	The name of a **Distributed Capture** profile to use.
`scannerlock=[true|false]`	Set to `true` to lock the scanner settings so that users cannot change them in Intelligent Capture Web Client. For example, if you do not want users to change the scanner settings that were set by a capture profile, then you would use this parameter. **Note:** Setting *`scannerlock`* to `true` does not reset existing scanner settings. Therefore, if a user has already changed scanner settings, then you might want the user to delete the entire browser cache to remove those settings. You can also select the **Scanner Lock** option for the **Distributed Capture** import profile in Intelligent Capture Designer.

For example:

```
http://localhost:8008/login.aspx?culture=zh-CN
http://localhost:8008/capture.aspx?culture=zh-CN
http://localhost:8008/Capture.aspx?profile=plain
http://localhost:8008/Capture.aspx?profile=plain&logoutret=http://www.example.com
http://localhost:8008/Capture.aspx?profile=plain&logoutret=http://
www.example.com&appvalues=name1:value1
```

4.8 Intelligent Capture Web Client Server-side Document Type Scripting

Intelligent Capture Web Client can use document type scripting, including both population and validation, on the REST server. However, because REST calls are short-lived and stateless, scripts called by REST calls do not have the `load`, `unload`, and the `DocumentExtracted` events.

For more information, see "Document Type Scripts" on page 42.

The following restrictions apply:

- Only the following events are supported. Because these events will be frequent, the scripts should not perform resource-intensive operations in them.

REST Service	Script Events
Extraction services (extract documents and pages; classify and extract documents and pages)	– `ExecutePopulationRule` – `ExecuteTableRowPopulationRule` – `ExecuteValidationRule` – `ExecuteTableRowValidationRule`
Populate	– `ExecutePopulationRule` – `ExecuteTableRowPopulationRule`
Validate	– `ExecuteValidationRule` – `ExecuteTableRowValidationRule`
Populate and validate	– `ExecutePopulationRule` – `ExecuteTableRowPopulationRule` – `ExecuteValidationRule` – `ExecuteTableRowValidationRule`

- The scripts cannot perform the following actions:
  - Access task values
  - Use `TaskFinishOnErrorNotAllowed`
  - Access a step's *`CustomValue`*
  - Change a document's locale and have that reflected on the client
  - Change the handling of question marks
  - Change whether operators can edit fields in a section, add, or delete rows
  - Hide fields
  - Disable validation rules

Chapter 5

Profile and Task Scripting API Reference

Profile and task scripting interfaces are provided as .NET libraries. Custom code is written using standard .NET programming languages such as Visual Basic and Visual C#.

Namespaces

- Emc.InputAccel.CaptureClient
- Emc.InputAccel.CaptureFlow
- Emc.InputAccel.ImageFilter
- Emc.InputAccel.UimScript

Chapter 6

Recognition Scripting

This guide provides detailed information and instructions for customizing Advanced Recognition modules to suit specific project requirements. Advanced Recognition functionality features a complete integrated development environment that includes:

- Dispatcher Function Library
- Dispatcher Event Model
- Dispatcher Object Model

Notes

- The following modules are now categorized as Advanced Recognition modules, which require an Advanced Recognition license:
  - Classification
  - Collector
  - Production Auto-Learning Supervisor (*PAL*)
- Dispatcher Manager functionality is now integrated into Intelligent Capture Designer and available in the Recognition area and called Recognition Designer.
- The Recognition module is deprecated and replaced with Extraction. The Classification Edit module is deprecated and replaced with Identification. The Validation module is deprecated and replaced with Completion. These deprecated modules can still be used and references to the modules exist in this documentation.

This section provides detailed information on the following:

- Writing good VBA code: Provides a Recognition project developer with the information necessary to be able to write good *VBA* code. Includes a matrix of object model elements.
- VBA Script Editor: Provides a graphical interface to the *IDE* (integrated development environment) for easy script editing.
- Dispatcher Function Library: A comprehensive library of functions for programming in the *IDE*.
- Dispatcher Event Model: Enables *VBA* scripts to respond to Recognition events and thus to modify batch data and objects to customize the classification and recognition steps.

- Dispatcher Object Model: The Dispatcher Object Model is a set of objects that enable access from a *VBA* script to Dispatcher objects, such as projects, batches, folders, documents and fields.
- Matrix tables: Comprehensive tables of Dispatcher objects, events, methods and their properties, showing how they are linked to each other.

6.1 What is VBA?

VBA (Visual Basic for Applications) is a high-level scripting language that is used for application-specific programming. It may be used as a general-purpose programming language or it may be limited to specific functions used to enhance the running of an application or program.

The *VBA* used in Advanced Recognition adds a VBA-compatible VBA Script Editor and debugger to your application, enabling the language to be extended with user-defined statements, thus enabling end-users to control their applications. It provides a complete integrated development technology, ideal for rapid customization and integration purposes.

VBA offers numerous advantages of industry standard programming including:

- A wide choice of functions and data types with the possibility to create user-defined functions
- Sharing of function libraries between scripts
- COM native interface enabling easy interfacing with external components
- Event driven programming schema
- Reusable code
- Rapid execution time
- Support centers, Web sites and easily accessible user guides

By default, Advanced Recognition use VB-COM. In scripting windows, users can specify either VB.NET or VB-COM. On the first line of a script, `#Language "WWB-COM"` indicates that VB-COM is activated with some enhancements. Change the line to `#Language "WWB.NET"` to activate VB.NET. Also, if no line starts with `#Language`, VB-COM is activated without `WWB-COM` enhancements. There are differences between the two languages, and VB.NET offers more possibilities than VB.COM. Also, types and available methods can differ between the two. The WinWrap documentation, accessed from the **Help** menu in the script editor, provides more information about scripting options.

6.2 Writing Good VBA Code

The information in this section enables a Dispatcher project developer to write good *VBA* code, to complete and enhance code recently converted to *VBA*, and to optimize it for Advanced Recognition. In addition, this section introduces and provides examples of new *VBA* coding possibilities in Advanced Recognition.

6.2.1 Using Methods

This section provides descriptions and examples of the methods used in Advanced Recognition.

6.2.1.1 AddRow

Adds a user row in a `DpDocArray` on the *`CurrentDocument`*. The user row is inserted in the *`RowNumber`* position. If the *`RowNumber`* is greater than the number of rows, adds the row to the table.

Generates a bad context error in three cases:

- If used somewhere else other than in the `CurrentDocument`.
- If used in the Classification module.
- If used in the Recognition module somewhere else other than in the event `AfterDocumentRecognition`.

Example: `CurrentDocument.Arrays(0).AddRow(0)`

PARAMETER:

`in RowNumber: Long`

6.2.1.2 AssignDocField

Copies the content of the `DpDocField` source object into this object

Example:

```
CurrentField.AssignDocField(CurrentFolder.Documents(1).Fields("Field1"))
```

PARAMETER:

`Source: DpDocField`

6.2.1.3 DeleteRow

Deletes the row in the *RowNumber* position in a DpDocArray on the *CurrentDocument*, if the position is correct (returns True). Returns False if the row position is not correct.

Generates a bad context error in three cases:

- If used somewhere else other than in the *CurrentDocument*.
- If used in the Classification module.
- If used in the Recognition module somewhere else other than in the AfterDocumentRecognition event or ControlDocument event.

Note: In some events, the value of the *CurrentField* property may be set to a specific value at the beginning of the script. When deleting the current row in a table, the value of the *CurrentField* property is set to Nothing.

Example:

```
CurrentDocument.Arrays(0).DeleteRow(0)
```

PARAMETERS:

in RowNumber: Long

out Result: Boolean

6.2.1.4 GetParamValue

Returns the current value of the field parameter *ParamName*. Initially, in the *CurrentDocument* the parameter has the default value defined in the associated template.

Generates a bad context error if used somewhere other than in the *CurrentDocument*.

Example:

```
Private Sub IndexingFamily_BeforeDocumentRecognition()
     Dim Temp As String
     Temp = CurrentDocument.Fields("Field1").GetParamValue("ParamName")
     CurrentDocument.Fields("Field2").Value = Temp
     CurrentDocument.Fields("Field2").SetStatusOK
End Sub
```

PARAMETERS:

ParamName: string

Return: String

6.2.1.5 GetPreviousDocument

Returns the previous document in the batch, if it exists. Nothing otherwise.

Example:

```
Dim Prev_Doc As DpDocument Set Prev_Doc = CurrentDocument.GetPreviousDocument
```

PARAMETER:

Return: DpDocument

6.2.1.6 GetNextDocument

Returns the next document in the batch if any. Nothing otherwise.

Example:

```
Dim Next_Doc As DpDocument Set Next_Doc = CurrentDocument.GetNextDocument
```

PARAMETER:

Return: DpDocument

6.2.1.7 GetSecondaryValue

Returns a secondary row value, for a primary row whose row number is *RowNumber*. The *Direction* parameter indicates the search direction. The secondary row position is given by the parameter *Position*.

The primary *Row* value is part of the value set. It may be returned as a Value.

Note: For a given primary line, the method `GetSecondaryValue` returns the value of a secondary line and `GetSecondaryValueCount` returns the number of secondary lines.

Example:

```
Dim i As Integer
Dim field As DpDocField
Dim Array As DpDocArray
Set Array=CurrentDocument.Arrays.Item(0)
  For i=0 To Array.RowCount-1
    If Array("Description").GetSecondaryValueCount(i,sdBelow)>=2 Then
    Set field=Array("Description").GetSecondaryValue(i,sdBelow,1)
    Array("Batch").Item(i).AssignDocField(field)
  End If
Next i
```

PARAMETERS:

RowNumber: Long

Direction: ESearchDirection

Position: Long

Return: DpDocField

6.2.1.8 GetPreviousFolder

Returns the previous folder of *CurrentFolder* if any; set to Nothing otherwise.

PARAMETERS:

Return: DpFolder

6.2.1.9 GetNextFolder

Returns the next folder of *CurrentFolder* if any; set to Nothing otherwise.

PARAMETER:

Return: DpFolder

6.2.1.10 GetSecondaryValueCount

Returns the secondary values count for a primary row whose row number is *RowNumber*. The *Direction* parameter indicates the search direction.

The primary *Row* is part of the value set.

Note: For a given primary line, the method `GetSecondaryValue` returns the value of a secondary line and `GetSecondaryValueCount` returns the number of secondary lines.

Example:

```
Dim i As Integer
Dim field As DpDocField
Dim Array As DpDocArray
Set Array=CurrentDocument.Arrays.Item(0)
For i=0 To Array.RowCount-1
    If Array("Description").GetSecondaryValueCount(i,sdBelow)>=2 Then
         Set field=Array("Description").GetSecondaryValue(i,sdBelow,1)
         Array("Batch").Item(i).AssignDocField(field)
    End If
Next i
```

PARAMETERS:

RowNumber: Long

Direction: ESearchDirection

Return: Long

6.2.1.11 GetTemplateByID

Returns the template whose ID is given by the parameter *ID*.

Example:

```
Set Template=CurrentProject.Templates.GetTemplateByID(1)
CurrentDocument.Fields("Champ2").Value =
Template.Name
```

PARAMETERS:

ID: Integer

Return: DpTemplate

6.2.1.12 IsParagraphHeader

Returns True if the line *RowNumber* is a paragraph header, False otherwise.

Example:

```
Row=0
For Each Field In CurrentDocument.Arrays(0)("Field1")
    Header = CurrentDocument.Arrays(0).IsParagraphHeader(Row)
    CurrentDocument.Arrays(0)("Field1")(Row).Value= CStr(Header)
    Row=Row+1
Next
```

PARAMETERS:

in RowNumber: Long

Return: Boolean

6.2.1.13 IsUserRow

Returns True if the line <*RowNumber*> was added by the user, False otherwise.

Example:

```
Row=0
For Each Field In CurrentDocument.Arrays(0)("Field1")
    CurrentDocument.Arrays(0)("Field1")(Row).Value= CStr(UserRow)
    Row=Row+1
Next
```

PARAMETERS:

in RowNumber: Long

Return: Boolean

6.2.1.14 RemoveCharacter

Deletes the character in the Index+1 position in a `DpDocField`, if the position is correct (returns True). Returns False if the position is not correct.

This method enables deleting a character and the associated character level information (which includes *`Bounds`* and *`Confidence (DpDocField)`* object properties).

The function `Replace` also enables deleting a character but as it does not delete the associated character level information it deactivates the character validation functionality for fields on which the **Character validation** is enabled in Recognition Designer. It is thus recommended to use the method `RemoveCharacter` in place of the function `Replace` to delete a character for fields on which the **Character validation** is enabled.

Example:

```
CurrentField.RemoveCharacter(0)
```

PARAMETERS:

`in Index: Long`

`out Result: Boolean`

6.2.1.15 RequiresValidation

When set to True, the value of the field requires validation by the user.

If the field has been defined as requiring systematic confirmation, the attribute is set to True, which means the user must confirm the field value manually.

PARAMETERS:

None

6.2.1.16 SetBounds

Sets the bounds of a `DpDocfield`. The bounding box coordinates are defined by its top left point (X,Y), its Width (W) and its Heigth (H). X,Y,W and H are in pixels. Refer to Setting focus on a field when inserting a table line to see illustrations of the `SetBounds` method.

Example:

```
CurrentDocument.Fields("Field1").SetBounds(0,50,200,50)
```

PARAMETERS:

`in X: Long`

`in Y: Long`

in W: Long

in H: Long

6.2.1.17 SetFocus

Sets the focus on a `DpDocField` (defined in the parameters). The cursor is placed in the field input box and the field is zoomed on the image if it is a placed field. Refer to Setting focus on a field when inserting a table line to see illustrations of the `SetFocus` method.

Generates a bad context error in three cases:

- If used somewhere else other than in the *`CurrentDocument`*.
- If used somewhere else other than in the `<Table>_AfterAddRow`, `<Table>_AfterDeleteRow` or `IndexingFamily_PressCustomHotKey` events.

Example: For an index field:

```
CurrentDocument.Fields("Field1").SetFocus
```

Example: For a table field:

```
CurrentDocument.Arrays("Mainarray")(0)(0).SetFocus
```

PARAMETERS:

None

6.2.1.18 SetParamValue

Sets the field parameter *`ParamName`* as the current value (*`ParamValue`*).

Example:

```
Private Sub IndexingFamily_AfterDocumentRecognition()
    Dim ParamValue As String
    ParamValue = " TEST    " + Str(CurrentDocument.Template.ID)
    CurrentDocument.Fields("Field1").SetParamValue("ParamName", ParamValue)

CurrentDocument.Fields("Field1").Value=CurrentDocument.Fields("Field1").GetParamValue("Pa
ramName")
    CurrentDocument.Fields("Field1").SetStatusOK
End Sub
```

PARAMETERS:

`ParamName: string`

`ParamValue: string`

6.2.1.19 SetStatusOK

Sets the value of the *Status* property of a field to csOK.

Example:

```
CurrentDocument.Fields("Field1").SetStatusOK or CurrentField.SetStatusOK
```

PARAMETERS:

None

6.2.1.20 SetStatusError

Sets the value of the *Status* property of a field to csError.

Note: The OutputMessage in the index script must be set each time SetStatusError is used. OutputMessage is automatically emptied whenever the *Status* of the field changes to csOK.

The SetStatusError function should not be used on an invisible field. This function puts an invisible field in error, the user is not able to validate the field and the batch is not validated.

Example:

```
CurrentDocument.Fields("Field1").SetStatusError or CurrentField. SetStatusError
```

PARAMETERS:

None

6.2.1.21 SkipRecognition

Used to skip the recognition on the Document. Sets Recognized to True. Throws a bad context error if called in an event other than: Field_BeforeRecognition, Field_AfterRecognition, IndexingFamily_BeforeDocumentRecognition, or IndexingFamily_AfterDocumentRecognition.

Also used to skip recognition on TableFields and IndexFields. Can be called only once for a field. Sets Recognized to True. Throws a bad context error if called in an event other than: Field_BeforeRecognition, Field_AfterRecognition.

Example:

```
Private Sub IndexingFamily_BeforeDocumentRecognition()
    CurrentDocument.SkipRecogntion
End Sub
OR
Private Sub Field_BeforeRecognition()
    CurrentField.SkipRecognition
End Sub
```

PARAMETERS:

None

6.2.2 Examples Using VBA Scripts

This section contains several pertinent examples that show you how to use *VBA* scripts in Advanced Recognition to carry out specific actions.

6.2.2.1 Classifying a Document According to the Previous and Next Document

This example shows the "Filling holes" method, which consists of using a document script to automatically classify a document according to the classification type of documents that precede or follow the document in the batch.

This functionality is advantageous in the case of processing identical documents. If one document from a batch of identical documents has not been classified and it is preceded or followed by documents that have been classified, then it is classified using a specific template script.

In this example, to classify the document, it is necessary to access to the information in the current document and the document that follows it. You cannot use the `Project_AfterDocumentClassification` event since it is called by the document and information contained in the document that follows is not yet available. In fact, when this event is called, for the second document in a batch, the third document has not been processed yet and its type is unknown. This control is called when all the documents have been classified, that is, on the event `Project_AfterClassification`.

Example:

```
Private Sub Project_AfterClassification()
    'Variables declaration
    Dim Folder As DpFolder
    Dim Doc As DpDocument
    'Loop in all folders of the batch
    For Each Folder In CurrentBatch.Folders
         'Loop in all documents of the folder
         For Each Doc In Folder.Documents
                Set prev_doc=Doc.GetPreviousDocument
                Set next_doc=Doc.GetNextDocument
                'If there is a document before and after the current document in the
folder and the template is the same for both,
       'then the current document is classified like the previous and next document.
                If TypeName(prev_doc) <> "Nothing" Then
                    If TypeName(next_doc) <> "Nothing" Then
                         If prev_doc.Classified =True Then
                                If next_doc.Classified =True Then
                                    If prev_doc.Template.Name =
next_doc.Template.Name Then
                                         Doc.Template=prev_doc.Template
                                    End If
                                End If
                         End If
                    End If
                End If
         Next
```

```
    Next
End Sub
```

6.2.2.2 Declassifying a Document

This section details declassifying a document, which is useful in preventing combinations of templates.

All actions concerning a document that has just been classified, as in the example below, should be run in the `Project_AfterDocumentClassification` event.

Example:

```
Private Sub Project_AfterDocumentClassification()
    If CurrentDocument.Classified = True Then
         If CurrentDocument.Template.Name = "Check" Then
                CurrentDocument.Template=Nothing
         End If
         End If
    End If
    End If
End Sub
```

6.2.2.3 Manual Classification of Documents That Have a Template Hypothesis

Use manual classification to assign a document to a matching template instead of using a default template. Manual classification applies when:

- Documents were not classified automatically.
- Unclassified documents may have what is called a *`TemplateHypotheses`*, that is, classification has managed to establish a certain link between the non-classified document and one or more templates in the project.

Since the choice of sending to manual classification is dependant on the classification of each document, it is necessary to call the event `Project_AfterDocumentClassification`.

Example:

```
Private Sub Project_AfterDocumentClassification()
         'Check if there is one or more template hypothesis
         If CurrentDocument.TemplateHypotheses.Count>0 Then
                'Force to go to manual classification
                CurrentDocument.Template=Nothing
         Else
                 'Affect a default template
                    CurrentDocument.Template=CurrentProject.Templates("Default")
         End If
    End If
End Sub
```

6.2.2.4 Accessing the Field Value Located in a Different Folder Document

Use the parameter *RankInFolder* of the searched document to manipulate the value of a field present in a document (other than the current document) in a folder.

Example:

```
Private Sub IndexingFamily_AfterDocumentRecognition()
    'Affectation value of field 1 of document 2 to field 1 of document 1

CurrentFolder.Documents(0).Fields("Field1").Value=CurrentFolder.Documents(1).Fields("Fiel
d1").Value
End Sub
```

6.2.2.5 Accessing the Field Value Located in a Different Batch Folder

When a searched field is located in the batch instead of the document of a folder, it is necessary to provide the precise folder in which the document is located.

Example:

```
Private Sub Field1_AfterRecognition()
    Dim Doc1 As DpDocument
    Dim Doc2 As DpDocument
    'Access to document 1 of folder 1 and of document 2 of folder 2
    Set Doc1=CurrentBatch.Folders(0).Documents(0)
    Set Doc2=CurrentBatch.Folders(1).Documents(1)
    'Assigning the value of field 1 of document 2 of folder 2 of field 1 of document 1
of folder 1
    Doc1.Fields("Field1").Value=Doc2.Fields("Field1").Value
End Sub
```

6.2.2.6 Bypassing the Automatic Classification of a Document

You assign a template to a document using the following script to avoid passing the document through automatic classification.

This can be useful when processing a succession of repetitive documents such as, account information statements and checks. By using a script, you can automatically classify the second document when the first document was classified with a template by Advanced Recognition.

When you want to attribute a template to a document before classification, call the `Project_BeforeDocumentClassification` event before classification of the current document.

Example:

```
Private Sub Project_BeforeDocumentClassification()
    'Check that there is a previous document
    If TypeName(CurrentDocument.GetPreviousDocument) <> "Nothing" Then
        'If the previous document has been classified as a bank statement
   'then the current document is classified as a check.
        If CurrentDocument.GetPreviousDocument.Classified Then
            If CurrentDocument.GetPreviousDocument.Template.Name="BankStatement" Then
                CurrentDocument.Template=CurrentProject.Templates("Check")
            End If
        End If
```

```
    End If
End Sub
```

6.2.2.7 Managing Intra-Field Index Controls

VBA runs by executing successive events that are triggered according to the state of the currently active batch. You need to know the status of the batch to perform field edits, such as when to verify the accuracy of the field data.

It is recommended that you use the same field check code in the events `IndexingFamily_ControlDocument` and `<Field Name>_ValidateValue`. The following example tests that field "A" is greater than or equal to 1.

Example:

```
'Creation of a generic function that is used in other events
Private Sub CheckA(Field As DpDocField)
    If Val(Field.Value)>=1 Then
         Field.SetStatusOK
    Else
         Field.SetStatusError
         Field.OutputMessage="The Value should be greater than 1"
    End If
End Sub
 '*********************************************************************
'This event is triggered when the user has validated the field manually
Private Sub A_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
  ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    'Call of the generic function to check the field A value
    CheckA(CurrentField)
End Sub
'*********************************************************************
'This event is triggered each time a whole control should be done on the document
Private Sub IndexingFamily_ControlDocument()
    'Call of the generic function to check the field A value
    CheckA(CurrentDocument.Fields("A"))
End Sub
```

6.2.2.8 Managing a Table Intra-Field Control

Managing a table intra-field control differs from that of an index field in the sense that each cell in the table field must be verified. Otherwise, the functionality is the same.

The following example verifies that each "quantity" line has a positive value. If this is not the case, the value is reinitialized to `0`.

Example:

```
'Creation of a generic function
Private Sub CheckQty(Field As DpDocField)
    If Val(Field.Value)<0 Then
         Field.Value="0"
    End if
End Sub
 '*********************************************************************
'This event is triggered each time a whole control is done on the document
Private Sub IndexingFamily_ControlDocument()
'Statement run only in recognition. Tests of table fields must not be called in
Classification step.
    If CurrentTask.TaskType=ttRecognition Then
         Dim Field As DpDocField
```

```
        'Loop in each cell of the table field
        For Each Field In CurrentDocument.Arrays.Item(0)("Qty")
        'Call of the generic function to check the field value
              CheckQty(Field)
        Next Field
    End If
End Sub
'*********************************************************************
'This event is triggered when the user has validated the field manually
Private Sub Qty_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
  ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    'Call of the generic function to check the field A value
    CheckQty(CurrentField)
End Sub
```

6.2.2.9 Managing an Index Inter-Field Control

It is common in all document types, that the integrity of one or more fields is interdependent. Consequently, it is necessary to ensure that a change in one or more field value does not negatively affect the others.

When interdependent fields are modified, you must verify the field value and its link to the other fields by calling the `ValidateValue` and the `IndexingFamily_ControlDocument` events on every field.

The following example checks that the zip code and the associated city/town correspond. The test is simplified and only works for Paris.

Example: Module 1:

```
Public Function CheckZipCode(ZipCode As String, City As String) As Boolean
    'This function checks that the zip code must be 75000 and city must be PARIS
    CheckZipCode=(ZipCode="75000" And City="PARIS")
End Function
```

Example: Module 2:

```
'Call of the module 1
'#Uses "*ZipCode.bas"
'Creation of a generic function that puts the field status to OK or KO depending on the
field values
Private Sub CheckZipCodeFields(ZipCodeField As DpDocField, CityField As DpDocField)
    If CheckZipCode(ZipCodeField.Value,CityField.Value) Then
         CityField.SetStatusOK
    Else
         CityField.SetStatusError
         CityField.OutputMessage="The zip code is incorrect"
    End If
End Sub
'*********************************************************************
Private Sub ZipCode_AfterRecognition()
    ' This is to simulate a wrong result from OCR
    CurrentField.Value="42800"
End Sub
'*********************************************************************
'This event is triggered every time a whole control is done on the document
Private Sub IndexingFamily_ControlDocument()
    CheckZipCodeFields(CurrentDocument.Fields("ZipCode"),CurrentDocument.Fields("City"))
End Sub
'*********************************************************************
'This event is triggered when the user has validated the field manually
Private Sub City_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
   ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
```

```
    CheckZipCodeFields(CurrentDocument.Fields("ZipCode"),CurrentField)
End Sub
'*********************************************************************
'This event is triggered when the user has validated the field manually
Private Sub ZipCode_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
 ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    CheckZipCodeFields(CurrentField,CurrentDocument.Fields("City"))
End Sub
```

6.2.2.10 Managing a Table Inter-Field Control

The management of a table field control is similar to an index field, except that each table field can have several cells. This requires checking the value of each cell, line by line, column by column, as well as verifying the sum of these values.

The event call is the same as the index inter-field controls. The following example carries out a control line by checking that the quantity multiplied by the unit price is equal to the net price. It also describes a column control that adds all the net prices compared to the total calculated amount read from the document.

Example:

```
'Creation of a generic function used to check that the extended price is equal to the
quantity * the unit price
Private Sub CheckQtyByUnitPrice(Qty As DpDocField, UnitPrice As DpDocField, ExtdPrice As
DpDocField)
    If Val(Qty.Value)*Val(UnitPrice.Value)=Val(ExtdPrice.Value) Then
         ExtdPrice.SetStatusOK
    Else
         ExtdPrice.SetStatusError
         ExtdPrice.OutputMessage="Wrong Extd price"
    End If
End Sub
'*********************************************************************
'Creation of a generic function used to check that the total is equal to the sum of
extended prices
Private Sub CheckTotal(ExtdPrice As DpDocArrayField, Total As DpDocField)
    Dim Sum As Integer
    Dim field As DpDocField
    Sum=0
    'Loop in each cell of the table field
    For Each field In ExtdPrice
         Sum=Sum+Val(field.Value)
    Next field
    If Sum=Val(Total.Value) Then
         Total.SetStatusOK
    Else
         Total.SetStatusError
         Total.OutputMessage="Wrong Total, expect "&CStr(Sum)
    End If
End Sub
'*********************************************************************
'This event is triggered each time a whole control is done on the document
Private Sub IndexingFamily_ControlDocument()
    Dim i As Integer
    Dim Array As DpDocArray
    'Loop in each line of the table
    Set Array=CurrentDocument.Arrays(0)
    For i=0 To Array.RowCount-1

CheckQtyByUnitPrice(Array("Qty").Item(i),Array("UnitPrice").Item(i),Array("ExtdPrice").It
em(i))
    Next i
    CheckTotal(Array("ExtdPrice"),CurrentDocument.Fields("Total"))
End Sub
```

```
'*********************************************************************
'This event is triggered when the user has validated the field manually
Private Sub Qty_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    Dim Array As DpDocArray
    Set Array=CurrentDocument.Arrays(0)

CheckQtyByUnitPrice(Array("Qty").Item(Array.CurrentRow),Array("UnitPrice").Item(Array.Cur
rentRow),
 Array("ExtdPrice").Item(Array.CurrentRow))
End Sub
'*********************************************************************
 'This event is triggered when the user has validated the field manually
Private Sub UnitPrice_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
   ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    Dim Array As DpDocArray
    Set Array=CurrentDocument.Arrays(0)

CheckQtyByUnitPrice(Array("Qty").Item(Array.CurrentRow),Array("UnitPrice").Item(Array.Cur
rentRow),
 Array("ExtdPrice").Item(Array.CurrentRow))
End Sub
'*********************************************************************
'This event is triggered when the user has validated the field manually
Private Sub ExtdPrice_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
   ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    Dim Array As DpDocArray
    Set Array=CurrentDocument.Arrays(0)

CheckQtyByUnitPrice(Array("Qty").Item(Array.CurrentRow),Array("UnitPrice").Item(Array.Cur
rentRow),
 Array("ExtdPrice").Item(Array.CurrentRow))
    CheckTotal(CurrentDocument.Arrays(0)("ExtdPrice"),CurrentDocument.Fields("Total"))
End Sub
'*********************************************************************
'This event is triggered when the user has validated the field manually
Private Sub Total_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
   ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    CheckTotal(CurrentDocument.Arrays(0)("ExtdPrice"),CurrentDocument.Fields("Total"))
End Sub
```

6.2.2.11 Using the Next Field Command

You can position the focus on a field after the `ValidateValue` event runs.

If the parameter is defined as “Nothing”, then the next field is determined.

The example below puts the focus on Field3 of the next document when the validated field has an error, when the value has not been modified by the user, and the control is called by pressing **ENTER**.

Example:

```
Private Sub Field1_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
    ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    If Validate and PreviousValue<>CurrentField.Value and CurrentField.Status=csError
Then
            If TypeName(CurrentDocument.GetNextDocument)<>"Nothing" Then
                'Next field command = Field3
                Set NextField = CurrentDocument.GetNextDocument.Fields(“Field3”)
         End If
```

```
    End If
End Sub
```

6.2.2.12 Using an Inter-Document Control

The logic of processing documents in batches and folders offers many possibilities for inter-document relations when using documents such as, invoices, checks, and forms.

Use the event `IndexingFamily_ControlDocument` to check the expected behavior of each document according to its place within the folder (or in the batch). The following example verifies that the subtotal of all of the invoices in a folder corresponds to the total of the last invoice in the folder.

Example:

```
'Creation of a generic function that makes the sum of each subtotal fields and
'compares that sum with the total field located in the last document of the folder.
Private Sub CheckSubTotal(Folder As DpFolder)
    Dim Sum As Integer
    Dim Total As Integer
    Dim Value As Integer
    Dim Doc As DpDocument
    Dim TotalField As DpDocField
    SetAllSubTotalFieldsToOK(Folder)
    Sum=0
    For Each Doc In Folder.Documents
         If Not Doc.Rejected Then
                If Doc.Template.Name="Invoice" Then
                    Sum=Sum+Val(Doc.Fields("SubTotal").Value)
                End If
         End If
    Next Doc
    'The last document of the folder should contain the Total
    Set Doc=Folder.Documents.Item(Folder.Documents.Count-1)
    If Doc.Template.Name="Invoice" Then
         Set TotalField=Doc.Fields("Total")
         Total=Val(TotalField.Value)
         If Total<>Sum Then
                TotalField.SetStatusError
                TotalField.OutputMessage="Wrong Sum of SubTotal fields, expect " &
CStr(Sum)
                For Each Doc In Folder.Documents
                    If Doc.Template.Name="Invoice" Then
                         Doc.Fields("SubTotal").SetStatusError
                         Value=Val(Doc.Fields("SubTotal").Value)
                         Doc.Fields("SubTotal").OutputMessage="Wrong Sum of SubTotal
fields, expect " & CStr(Total-(Sum-Value))
                    End If
                Next Doc
         End If
    End If
End Sub
'*********************************************************************
'Creation of a generic script, that sets all the total fields of the folder to OK
Private Sub SetAllSubTotalFieldsToOK(Folder As DpFolder)
    Dim Doc As DpDocument
    For Each Doc In Folder.Documents
         If Doc.Template.Name="Invoice" Then
                Doc.Fields("SubTotal").SetStatusOK
                Doc.Fields("Total").SetStatusOK
         End If
    Next Doc
End Sub
'*********************************************************************
'This event is triggered each time a whole control is done on the document
```

```
Private Sub IndexingFamily_ControlDocument()
    Dim last_document_in_folder As Boolean
    If CurrentTask.TaskType=ttRecognition Then
         'Sets the Total field to Read-Only if not the last invoice of the folder

last_document_in_folder=(CurrentDocument.RankInFolder=CurrentDocument.Folder.Documents.Co
unt-1)
         CurrentDocument.Fields("Total").ReadOnly=Not last_document_in_folder
         'Call of the generic CheckSubTotal function
         CheckSubTotal(CurrentFolder)
    End If
End Sub
'*********************************************************************
'This event is triggered when the user has validated the field manually
    If Validate And PreviousValue=CurrentField.Value And CurrentField.Status=csError Then
Private Sub SubTotal_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
    ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
         'Check that the current document is not the last of the folder
         If CurrentDocument.RankInFolder<CurrentFolder.Documents.Count-1 Then
     'Sets the next field as the SubTotal of the next document
              Set NextField=CurrentFolder.Documents.Item(CurrentDocument.RankInFolder
+1).Fields("SubTotal")
         Else
         'Sets the next field as the Total of the current document
               Set NextField=CurrentDocument.Fields("Total")
         End If
         CheckFieldFormat=False
         Exit Sub
    End If
    'Call of the generic CheckSubTotal function to check that the sum of the SubTotal
fields is equal to Total
    CheckSubTotal(CurrentFolder)
End Sub
'*********************************************************************
'This event is triggered when the user has validated the field manually
Private Sub Total_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
    ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    If Validate And PreviousValue=CurrentField.Value And CurrentField.Status=csError Then
        'Sets the next field to the SubTotal of the first document of the folder if the
Total field is wrong.
         Set NextField=CurrentFolder.Documents.Item(0).Fields("SubTotal")
         CheckFieldFormat=False
         Exit Sub
    End If
         'Call of the generic CheckSubTotal function to check that the sum of the
SubTotal fields is equal to Total
    CheckSubTotal(CurrentFolder)
End Sub
```

6.2.2.13 Inserting and Deleting Table Lines

In the scope of managing table lines, it is possible to add and delete lines using VBA script, a useful functionality when validating.

The following example enables a user, using the keyboard shortcut **CTRL + A**, to add a specific number of lines with the help of a window. The keyboard shortcut **CTRL + B** enables the user to delete the current line if it is empty. The keyboard shortcut **CTRL + C** deletes all the empty lines in a table. Moreover, after recognition, the `IndexingFamily_AfterDocumentRecognition` control also deletes all the empty lines in a table.

Example:

```
'Creation of a generic function that
makes the sum of each subtotal fields and compares that sum with the total
field located in the last document of the folder. Private Sub CheckSubTotal(Folder
As DpFolder)
    Dim Sum
As Integer
     Dim Total
As Integer
    Dim Value
As Integer
     Dim Doc
As DpDocument
    Dim
TotalField As DpDocField
    SetAllSubTotalFieldsToOK(Folder)
    Sum=0
    For
Each Doc In Folder.Documents
 If Not Doc.Rejected Then
        If Doc.Template.Name="Invoice" Then
            Sum=Sum+Val(Doc.Fields("SubTotal").Value
                End If
 End If
    Next Doc
    'The last document of the folder should contain the Total
    Set Doc=Folder.Documents.Item(Folder.Documents.Count-1)
    If Doc.Template.Name="Invoice" Then
         Set TotalField=Doc.Fields("Total")
         Total=Val(TotalField.Value)
         If Total<>Sum Then
                TotalField.SetStatusError
                TotalField.OutputMessage="Wrong Sum of SubTotal
fields, expect " & CStr(Sum)
        For Each Doc In Folder.Documents
            If Doc.Template.Name="Invoice" Then
                 Doc.Fields("SubTotal").SetStatusError
                        Value=Val(Doc.Fields("SubTotal").Value)
                        Doc.Fields("SubTotal").OutputMessage="Wrong Sum
of SubTotal fields, expect " & CStr(Total-(Sum-Value))
                    End If
        Next Doc
         End If
    End If
End
Sub
'*********************************************************************
'Creation of a generic script, that sets all the total
fields of the folder to OK
Private
Sub SetAllSubTotalFieldsToOK(Folder As DpFolder)
    Dim Doc As DpDocument
    For Each Doc In Folder.Documents
         If Doc.Template.Name="Invoice" Then
                Doc.Fields("SubTotal").SetStatusOK
                Doc.Fields("Total").SetStatusOK
         End If
    Next
Doc
End Sub
'*********************************************************************
'This event is triggered each time a whole control is
done on the document
Private
Sub IndexingFamily_ControlDocument()
    Dim
last_document_in_folder As Boolean
    If
CurrentTask.TaskType=ttRecognition Then

 'Sets the Total field to Read-Only if not the last invoice of the folder
```

```
last_document_in_folder=(CurrentDocument.RankInFolder=CurrentDocument.Folder.Documents.Co
unt-1)
         CurrentDocument.Fields("Total").ReadOnly=Not last_document_in_folder
         'Call of the generic CheckSubTotal function
         CheckSubTotal(CurrentFolder)
    End If
End
Sub
'*********************************************************************
'This event is triggered when the user
has validated the field manually
Private
Sub SubTotal_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    If
Validate And PreviousValue=CurrentField.Value And CurrentField.Status=csError
Then
         'Check that the current document is not the last of the folder
          If CurrentDocument.RankInFolder<CurrentFolder.Documents.Count-1
Then
         'Sets the next field as the SubTotal of the next document
                Set NextField=CurrentFolder.Documents.Item(CurrentDocument.RankInFolder
+1).Fields("SubTotal")
         Else
 'Sets the next field as the Total of the current document
                Set NextField=CurrentDocument.Fields("Total")
         End If
 CheckFieldFormat=False
 Exit Sub
    End If
    'Call of the generic CheckSubTotal function to check that the sum of the SubTotal
fields is equal to Total
    CheckSubTotal(CurrentFolder)
End Sub
'*********************************************************************
'This event is triggered when the user
has validated the field manually
Private
Sub Total_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    If
Validate And PreviousValue=CurrentField.Value And CurrentField.Status=csError
Then
        'Sets the next
field to the SubTotal of the first document of the folder if the Total field
is wrong.
         Set NextField=CurrentFolder.Documents.Item(0).Fields("SubTotal")
         CheckFieldFormat=False
         Exit Sub
    End
If
         'Call of
the generic CheckSubTotal function to check that the sum of the SubTotal fields
is equal to Total
    CheckSubTotal(CurrentFolder)
End Sub
```

6.2.2.14 Managing Secondary Lines

The method `AssignDocField` is used during secondary line controls and transfers information from a table field in a secondary line to a primary line to which it is linked. This function copies the value and the coordinates, as well as the field value.

Example:

```
Dim i As Integer
  Dim field As DpDocField
  Dim Array As DpDocArray
  Set Array=CurrentDocument.Arrays.Item(0)
 'Loop in all rows of the MainArray
  For i=0 To Array.RowCount-1
    'Get the first    secondary line below the primary line at index i
    Set field=Array("Description").GetSecondaryValue(i,sdBelow,1)
    'Puts in the Batch table field the value of the first secondary line in the
Description table field
    Array("Batch").Item(i).AssignDocField(field)
  Next i
```

6.2.2.15 Setting Focus on a Field When Inserting a Table Line

When inserting a table line, it is possible to place the cursor in one of the fields of the new created line and have a zoom on the field in the image.

`SetBounds` method can be combined with `SetFocus` method as in the following example. A customized hot key is defined to trigger a script that creates a table line and that applies `SetBounds` and `SetFocus` on one field set in the parameters. As a result, the cursor is placed in the field input box and the field is zoomed on the image.

Example:

```
Private Sub IndexingFamily_PressCustomHotKey(ByVal KeyCode As Long,
    ByVal Shift As Boolean, ByVal Alt As Boolean, ByVal Ctrl As Boolean, Handled As
Boolean, ByVal CurrenuRow As Long)
    Dim linenumber As Integer
    Dim rect As SRect
    Dim docfield As DpDocField
    If Ctrl And Not Shift And Not Alt Then
        If KeyCode=65 Then
            If True Then
                linenumber=CurrentDocument.Arrays("MainArray").CurrentRow
                CurrentDocument.Arrays("Mainarray").AddRow(linenumber+1)
                docfield=CurrentDocument.Arrays("Mainarray")("item")(linenumber)
                rect=docfield.Bounds
                CurrentDocument.Arrays("Mainarray")("item")(linenumber
+1).SetBounds(rect.X,rect.Y+20,rect.W,rect.H)
                CurrentDocument.Arrays("Mainarray")("item")(linenumber+1).Value="value"
                CurrentDocument.Arrays("Mainarray")("item")(linenumber+1).SetFocus
            End If
        End If
    End If
End Sub
```

6.2.3 Avoiding Difficulties When Using VBA

This section describes workarounds to help avoid difficulties when coding in *VBA*.

6.2.3.1 Using the Option Explicit Instruction

To avoid any corruption in the script, it is strongly advised to apply *Option Explicit* at the beginning of the code. This means that each variable must be declared before it is used, without which it would not be recognized as such.

6.2.3.2 Multiple Variable Declarations

Multiple variable declarations of the same type on one single line are possible with WinWrap *VBA*.

6.2.3.3 Using the Set Instruction to Affect an Object

When affecting an object value to a variable declared as an object, the Set instruction must be placed before the expression.

Example: *`Dim Field as DpDocField Set Field=CurrentDocument.Fields("A")`*

6.2.3.4 Task Type Test in ControlDocument

The event `IndexingFamily_ControlDocument` can be triggered during recognition, but also during automatic classification.

When running these modules, access to object values is different. For example, access to the field values of a table is not possible during classification.

This is why it is very important to define in the `IndexingFamily_ControlDocument` event the task in which you want to act.

Example:

```
If CurrentTask.TaskType=ttClassification then
    ' ... Test of pre-indexed fields
ElseIf CurrentTask.TaskType=ttRecognition then
    '
                                    … Test of not pre-indexed fields + table fields
End If
```

Note: The following tasks are also written in the same way: *`ttRecognition`*.

Remarks

If the template contains pre-index fields, a `controlDocument` event is run with `taskType=classification` just before another `controlDocument` event with `taskType=Recognition`.

6.2.3.5 Virtual Keys

Table 6-1: Virtual keys for Advanced Recognition

Decimal	Character
8	**BACKSPACE**
9	**TAB**
13	**ENTER**
16	**SHIFT** (both)
17	**CTRL** (both)
19	**PAUSE**
20	**CAPS LOCK**
27	**ESC**
32	**SPACEBAR**
33	**PAGE UP**
34	**PAGE DOWN**
35	**END**
36	**HOME**
37	(**LEFT ARROW**)
38	(**UP ARROW**)
39	(**RIGHT ARROW**)
40	(**DOWN ARROW**)
45	**INSERT**
46	**DELETE**
48	**0**
49	**1**
50	**2**
51	**3**
52	**4**
53	**5**
54	**6**
55	**7**
56	**8**
57	**9**
65	**A** or **A**

Decimal	Character
66	**B** or **B**
67	**C** or **C**
68	**D** or **D**
69	**E** or **E**
70	**F** or **F**
71	**G** or **G**
72	**H** or **H**
73	**I** or **I**
74	**J** or **J**
75	**K** or **K**
76	**L** or **L**
77	**M** or **M**
78	**N** or **N**
79	**O** or **O**
80	**P** or **P**
81	**Q** or **Q**
82	**R** or **R**
83	**S** or **S**
84	**T** or **T**
85	**U** or **U**
86	**V** or **V**
87	**W** or **W**
88	**X** or **X**
89	**Y** or **Y**
90	**Z** or **Z**
91	(**LEFT WINDOWS KEY**)
92	(**RIGHT WINDOWS KEY**)
93	(**APPLICATION KEY**: located between the right Windows and **CTRL** keys on most keyboards)
96	**0** (numeric keypad on the keyboard with Num Lock on)
97	**1** (numeric keypad on the keyboard with Num Lock on)

Decimal	Character
98	**2** (numeric keypad on the keyboard with Num Lock on)
99	**3** (numeric keypad on the keyboard with Num Lock on)
100	**4** (numeric keypad on the keyboard with Num Lock on)
101	**5** (numeric keypad on the keyboard with Num Lock on)
102	**6** (numeric keypad on the keyboard with Num Lock on)
103	**7** (numeric keypad on the keyboard with Num Lock on)
104	**8** (numeric keypad on the keyboard with Num Lock on)
105	**9** (numeric keypad on the keyboard with Num Lock on)
106	* (numeric keypad on the keyboard)
107	+ (numeric keypad on the keyboard)
109	- (numeric keypad on the keyboard)
110	. (numeric keypad on the keyboard)
111	/ (numeric keypad on the keyboard)
112	**F1**
113	**F2**
114	**F3**
115	**F4**
116	**F5**
117	**F6**
118	**F7**
119	**F8**
120	**F9**
122	**F11**
123	**F12**
144	**NUM LOCK**
145	**SCROLL LOCK**
186	;
187	=

Decimal	Character
188	,
189	-
190	.
191	/
192	`
219	[
220	\
221	]
222	'

6.2.3.6 Accessing the Current Values of a Script

Dispatcher Object Model access variables such as `CurrentTask`and `CurrentDocument` are only accessible in the project family script. Current values cannot be accessed by using an include script command from another script. The only solution to resolve this issue is to pass the searched value as a parameter in the called functions.

Example: Correct

```
MyFamily.bas
'#uses "Script1.bas"
Private Sub Project_ControlDocument()
    MyFunction(CurrentField)
End Sub
Script1.bas
Public Sub MyFunction(Field As DpDocField)
    MsgBox Field.Value
End Sub
```

Example: Incorrect

```
MyFamily.bas
'#uses "Script1.bas"
Private Sub Project_ControlDocument()
    MyFunction
End Sub
Script1.bas
Public Sub MyFunction
    MsgBox CurrentField.Value
End Sub
```

6.2.3.7 Global Variables and Unplaced Fields

If you require values that can be reused in different events within a module (for example the recognition step), you need to create global variables.

Several events are called during processing by a module but the values that are declared in one event are not accessible from another event within the same module. If you need a value that can be accessed from any event in the module, create a global variable. The best solution is to keep the value in an unplaced invisible field, then, you can use it in recognition.

6.2.3.8 Centralizing Field Names

The Dispatcher Object Model uses numerous strings to access objects that are relative to fields. For example, to obtain the field `InvoiceNumber` on the current document, the syntax is:

```
CurrentDocument.Fields("InvoiceNumber").Value
```

Since the string corresponding to the field name is used in many events, it is recommended to use a constant rather than a “hard-coded” string. This makes it easier to change the name of the field. Changing only the value of the constant is thus sufficient and avoids having to carry out a search and replace in all the inclusion scripts.

Example: Code Without Centralization of the Constant

```
Private Sub InvoiceNumber_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As
Boolean,
    ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
CurrentDocument.Fields("InvoiceNumber").Value = "Test"
End Sub
```

Example: Code With Centralization of the Constant

```
Public Const CTE_FIELDNAME_InvoiceNumber= "InvoiceNumber"
Private Sub A_ValidateValue(ByVal CurrentRow As Long, ByVal Validate As Boolean,
    ByVal PreviousValue As String, NextField As DispatcherObjectModel.DpDocField,
CheckFieldFormat As Boolean)
    CurrentDocument.Fields(CTE_FIELDNAME_InvoiceNumber).Value = "Test"
End Sub
```

6.3 VBA Script Editor

This section describes the graphical interface used to edit *VBA* scripts in Advanced Recognition.

Advanced Recognition is delivered with an integrated development environment (IDE) equipped with a graphical interface (the Script Editor) that is used to:

- Create and edit *VBA* scripts for objects and events
- Debug code
- Create user dialogs

- Import COM objects

6.3.1 Script File Format

Script files can be saved in the following formats.

Table 6-2: Script File Formats

Entity	Script file extension	Can be created in …
Project	BAS	Project
Module	BAS	Project or Index family
Object	OBJ	Project or Index family
Index family	BAS	Index family
Class	CLS	Project or Index family

The directory *<[Dispatcher project directory]>*`\Resources\Scripts` is used for project script files and script files that are used for sharing objects between modules of different indexing families.

Index family script files are saved by default in *<[Recognition project directory]>*`\IdxClasses`.

6.3.2 Using the VBA Script Editor

The *VBA* Script Editor can be opened in Recognition Designer from the menu **File** > **Edit Project Script** in the **Classification**, **Indexing**, and **Edit Index Families** views. The Script Editor cannot be opened if the project has been modified and has not been saved.

After a script file has been modified, it must be saved when closing the Script Editor. Saving the script file has no effect on the status of a Dispatcher project.

If a script file contains any syntax errors, it cannot be run. Any resulting error messages are visible in the status bar.

Before modifying a script, stop all the tests that may be running. If you stop a batch test, you cannot restart it. After the script has been saved, any modifications in it are taken into account the next time the batch is tested.

Note: The number of script files that can be opened in the Script Editor for editing at any given moment is limited to nine.

Help on using the *VBA* Script Editor is available from the Script Editor menu **?** > **Help on the Editor**. This guide contains detailed information on *VBA* functions as well as help on the editor menus, toolbars, and shortcut keys.

6.3.3 Shortcut Keys

Table 6-3: Script Editor Commenting Shortcut Keys

Key	Used to
CTRL + ALT + C;	Turn the current selection/line in the code into a comment
CTRL + ALT + U,	Delete the comment attribute for the current selection/line in the code
ALTGR + F5	Carry out a syntax check in the current edited basic project script and highlights in red the first error, if one exists

Table 6-4: Script Editor Editing Shortcut Keys

Key	Used to
TAB	Move selected lines right (Indent)
SHIFT + TAB	Move selected lines left (Outdent)
CTRL + F	Find a string
CTRL + R	Replace a string with another
F3	Repeat last find or replace (Again)
CTRL + SPACE	Complete the word
CTRL + I	Show the parameter information
CTRL + A	Activate the macro editing window
CTRL + E	Show the immediate output window
CTRL + W	Show the watch expressions window
CTRL + T	Show the call stack window
CTRL + L	Show the loaded macros/modules window
F5	Run the macro to completion
ESC	Stop the macro/module
F8	Run the current line
SHIFT + F8	Run the next line
CTRL + F8	Step out the current subroutine or function call
F7	Run until the line the cursor is on the current line
F9	Toggle the break point on the current line
SHIFT + CTRL + F9	Clear all break points

Key	Used to
SHIFT + F9	Show the value of the expression under of the cursor in the immediate window
CTRL-1...CTRL-9	Show this macro/module
SHIFT + F1	Show the table of contents for the WinWrap Basic Language
F1	Show WinWrap Basic language guide for the keyword under the cursor

Note: There is no shortcut for selecting the entire script.

6.3.4 Editing a Project Script File

Each Recognition project has one corresponding script, `DispatcherProject.bas`, which is created by default when the project is first created. This file can be opened in Recognition Designer in the Classification view and Indexing view.

6.3.5 Editing an Index Family Script File

Each index family has one corresponding script, `indexing_family_name.bas`, which is created by default when the index family is first created. This file can be opened in Recognition Designer in the **Edit indexing families** view.

6.3.6 Referencing Libraries

By default, all Advanced Recognition *VBA* script files reference two files:

- `DispatcherObjectModel.dll`
- `DispatcherFunctionLibrary.dll`

VBA objects created in the Script Editor are added to the Dispatcher Object Model library file and new functions are added to the Dispatcher Function Library file. You can also link to other library files from the **Edit** > **References** menu in the *VBA* Script Editor.

6.3.7 Using Module Scripts

VBA module scripts saved in the Scripts directory can be used:

- In another module script located in the same directory
- From the project script
- From module scripts belonging to an index family

The Uses clause #Uses "*module_name" (the module name is prefixed by the character *) enables a module script to be used by another module script. Here, the Uses clause is used as a relative path. It can also be used by an absolute path: `#Uses D:\Dispatcher\Document\Test\IdxClasses\Test.bas`.

Do not include a project script directly in an indexing family script. To perform functions common to project and index family scripts, write a separate script and save it to the `\Resources\Scripts\` folder. Then include the script in a project or index family script with the `#Uses` command.

Note: Unless otherwise specified, module script files which have the extension .bas are searched for by default.

Indexation tests can be run from the Indexing view; events that are linked to objects in the indexation template are run according to how they are defined in the *VBA* script modules.

Word breaking points

Word breaking points are defined in the *VBA* Script Editor and are saved automatically in the file `DispatcherProject.dbp` in the directory *<[Recognition project directory]>*`\Resources\Scripts` when the break points are defined in the project script. They are saved in the `Indexing_Family_Name.dbp` file in the *<[Recognition project]>*`\IdxClasses` directory when the break points are defined in the family script. When a point in the script module predefined as being a break point is found, the *VBA* Script Editor opens with the corresponding code displayed. Break points are run only on unit tests and template tests if the debug mode has been selected.

6.3.8 Using Breakpoints

This topic applies to Advanced Recognition; it does not apply to Dispatcher Stand-Alone.

Breakpoints enable you to look for bugs through your code. A breakpoint is a point that tells the debugger to temporarily suspend the execution of your program. Execution can be restarted at any time. This section provides you information on saving breakpoints in a project script.

To set a breakpoint in the script editor:

1. Open Advanced Recognition.

2. Open a project or an index family script window.

   If you want to create or edit a project script, select the menu **File > Edit Project Script**. The **Edit Project Script** window appears. The name of the script file corresponds to the project (plus .bas extension), and is saved to the `[Project directory]\Resources\Scripts` directory.

   If you want to create or edit an index family script, select the menu **File > Edit Family Script**. The **Edit Family Script** window appears. The name of the script file corresponds to the index family (plus .bas extension), and is saved to the `[Project directory]\Resources\Scripts\IdxClasses` directory.

3. When you open the .dpp file in Recognition Designer and you use a UNC path, make sure you also use a UNC path to open the .dpp file from any of the Advanced Recognition modules in setup mode, otherwise the script debug mode will not work. Also, When you open the .dpp file in Recognition Designer and you use a local path, make sure you also use a local path to open the .dpp file from any of the Advanced Recognition modules in setup mode, otherwise the script debug mode will not work

   Caution

   If you copy or move your Recognition project to another directory, you must redefine your breakpoints since they are specific to the Recognition project path.

4. In your script, move to the line of code where you want to set a breakpoint.

5. To set a breakpoint, select the **Breakpoint** option from the **Debug** menu.

   A breakpoint is effective only if it is placed on a line of executable code. Breakpoints that are placed on comments, blank lines, or any other lines that are not executable code are not taken into account and will not act as breakpoints.

To run a script in debug mode:

1. Run an Advanced Recognition module from a command prompt in production, using the `-scriptdebug` argument.

   Note: The `-scriptdebug` command line argument cannot be used when Advanced Recognition modules are run as services.

2. From the production window, run a batch. The process stops as soon as a breakpoint is reached and the **Edit Project Script** window appears. You cannot edit a script while executing it. Scripts can only be edited from Recognition Designer.

6.3.9 Running Tests

Index family script routines can be tested in the **Edit Indexing Families** window from a unit test on a field. For this to work, one sub-main routine must be defined in which the event that requires testing is called.

Note: The sub-main routine can be defined only once in the project.

6.4 Dispatcher Function Library

The Dispatcher Function Library (*DFL*) functions were used exclusively in Dispatcher releases 4.6 and earlier. These functions are identified by the prefix *DFL*. The *DFL* functions are available for use in all existing or new scripts. A *DFL* function is called simply by typing the function name in the script.

However, VBA is the current scripting language used and should be substituted for *DFL* functions whenever possible. It is a best practice to use *VBA* functions rather than Dispatcher library functions.

Dispatcher library functions are grouped into different types, which are explored in this section.

6.4.1 DFL Functions Not Supporting Surrogate Pairs

Surrogate pairs are sequences of two 16-bit Unicode encoding values that represent a single character. The first value is the high surrogate and ranges from U+D800 to U +DBFF. The second value is the low surrogate and ranges from U+DC00 to U+DFFF. For information on surrogate pairs, refer to the Microsoft *MSDN* web site or the Unicode web site and search on surrogate pairs.

These *DFL* functions do *not* support surrogate pairs:

- "`DFLCompareStrings`" on page 179
- "`DFLCompareStringsEx`" on page 180
- "`DFLPos`" on page 191
- "`DFLRegExp`" on page 192
- "`DFLSetLength`" on page 196
- "`DFLStrCopy`" on page 197
- "`DFLStrNSet`" on page 199
- "`DFLStrSet`" on page 199

6.4.2 DFL Mathematical Functions

Use the equivalent *VBA* function instead of the Dispatcher library function whenever possible.

Table 6-5: Dispatcher Mathematical Functions and VBA Equivalents

Function	VBA equivalent
DFLArccos	-
DFLArcsin	-
DFLArctan	Atn (D) : D
DFLFloor	-
DFLLn	Log(D) : D
DFLLog2	-
DFLLog10	-
DFLLogN	-
DFLModulo	-
DFLPower	-
DFLRandom	Rnd(I) :D
DFLSqr	-
DFLSqrt	Sqr(D)
DFLTrunc	-

6.4.2.1 DFLArccos

Syntax

Visual Basic

```
DFLArcCos(Num As Double) As Double
```

Details

Return Value: Num

The inverse cosine of the valueNum expressed as a radian.

Table 6-6: Parameters

Parameter	Description
Num As Double	Numeric value expressed in radians

Example:

```
DFLArcCos(0.9) -> 0.451026811796262
```

6.4.2.2 DFLArcsin

Return Value: Num

The inverse sine of the value Num expressed as a radian.

Syntax

Visual Basic

```
DFLArcSin(Num As Double) As Double
```

Details

Table 6-7: Parameters

Parameter	Description
Num	Numeric value expressed in radians

Example:

```
DFLArcSin(0.9) -> 1.11976951499863
```

6.4.2.3 DFLArctan

Return Value: Num

The inverse tangent of the value Num expressed as a radian.

Syntax

Visual Basic

```
DFLArTan(Num As Double) As Double
```

Details

Table 6-8: Parameters

Parameter	Description
Num	Numeric value expressed in radians

Example:

```
DFLArcTan(0.9) -> .732815101786507
```

6.4.2.4 DFLFloor

Return Value: Num
The next lower Integer value (as a floating point number) from the argument Num.

Syntax

Visual Basic

```
DFLFloor(Num As Double) As Double
```

Details

Table 6-9: Parameters

Parameter	Description
Num	Numeric value

Example:

```
DFLFloor( -11.9 ) -> -12
```

```
DFLFloor( 12.1 ) -> 12
```

6.4.2.5 DFLLN

Return Value: Num
The neperian logarithm of a positive number Num.

Syntax

Visual Basic

```
DFLLn(Num As Double) As Double
```

Details

Table 6-10: Parameters

Parameter	Description
Num As Double	Numeric value

Example:

```
DFLLn(3) -> 1.09861228866811
```

6.4.2.6 DFLLog2

Return Value: Num
The base 2 logarithm of a positive number Num.

Syntax

Visual Basic

```
DFLLog2(Num As Double) As Double
```

Details

Table 6-11: Parameters

Parameter	Description
Num As Double	Numeric value

Example:

```
DFLLog2(8) -> 3
```

6.4.2.7 DFLLog10

Return Value: Num
The decimal logarithm of a positive number Num.

Syntax

Visual Basic

```
DFLLog10(Num As Double) As Double
```

Details

Table 6-12: Parameters

Parameter	Description
Num As Double	Numeric value

Example:

```
DFLLog2(1000) -> 3
```

6.4.2.8 DFLLogN

Return Value: Num
The base logarithm of a positive number Num.

Syntax

Visual Basic

```
DFLLogN(Base As Double, Num As Double) As Double
```

Details

Table 6-13: Parameters

Parameter	Description
Base As Double	Base value
Num As Double	Numeric value

Example:

```
DFLLogN(5, 25) -> 2
```

6.4.2.9 DFLModulo

Return Value: Num
Num1 modulo Num2, which is the rest of a division of Num1 by Num2.

Syntax

Visual Basic

```
DFLModulo(Num1 As Long, Num2 As Long) As Long
```

Details

Table 6-14: Parameters

Parameter	Description
Num1 As Double	Dividend
Num2 As Double	Divisor

Example:

```
DFLModulo( 153 , 15 ) -> 3
```

Caution
If Num2=0, an error is returned.

6.4.2.10 DLFPower

Return Value: Num, Pow
The first number Num raised to the second number Pow.

Syntax

Visual Basic

```
DFLPower(Num As Double, Pow As Double) As Double
```

Details

Table 6-15: Parameters

Parameter	Description
Num As Double	Numeric value
Pow As Double	Exponent value

Example:

```
DFLPower(2,4) -> 16
```

6.4.2.11 DFLRandom

Return Value: Num
A random number within a specified range between 0 (included) and Num (excluded) parameters .

Syntax

Visual Basic

```
DFLRandom(Num As Long) As Long
```

Details

Table 6-16: Parameters

Parameter	Description
Num As Double	Range boundary

6.4.2.12 DFLSqr

Return Value: Num
The square of the argument Num, which must be a positive number.

Syntax

Visual Basic

```
DFLSqr(Num As Double) As Double
```

Details

Table 6-17: Parameters

Parameter	Description
Num As Double	Numeric value

Example:

```
DFLSqr(-4) -> 16
```

```
DFLSqr(2.5) -> 6.25
```

6.4.2.13 DFLSqRt

Return Value: Num
The square root of the argument Num, which must be a positive number.

Syntax

Visual Basic

```
DFLSqRt(Num As Double) As Double
```

Details

Table 6-18: Parameters

Parameter	Description
Num As Double	Positive Numeric Value

Example:

```
DFLSqRt( -4 ) -> Error
```

```
DFLSqRt( 16.0 ) -> 4.0
```

6.4.2.14 DFLTrunc

Return Value: Num

The truncated Integer Value (as a floating point number) from the argument Num.

Syntax

Visual Basic

```
DFLTrunc(Num As Double) As Double
```

Details

Table 6-19: Parameters

Parameter	Description
Num As Double	Numeric Value

Example:

```
DFLTrunc( -11.9 ) -> -11.0
DFLTrunc( 12.1 ) -> 12.0
```

6.4.3 DFL String Functions

Use the equivalent *VBA* function instead of the Dispatcher library function whenever possible.

Table 6-20: String Functions and *VBA* Equivalents

Function	VBA equivalent
DFLAddQuote	-
DFLASCII	Asc(S): I
DFLCharOccurenceCount	-
DFLCheckFormat	-
DFLCommaText	-
DFLCommonCharCount	-
DFLCommonNLeft	-
DFLCommonNRight	-
DFLCompareStrings	-
DFLCompareStringsEx	-
DFLFloatToStr	CStr(D): S

Function	VBA equivalent
DFLFloatToStrF	-
DFLFormatAmount	-
DFLFormatVal	-
DFLGetLength	Len(S): I
DFLIntToStr	CStr(I):S
DFLIsAmount	-
DFLIsInteger	IsNumeric(S) : B
DFLKeepChar	-
DFLLeftAlign	-
DFLLeftAlignChar	-
DFLLeftAlignNum	-
DFLPos	InStr (S,S): I
DFLRegExp	-
DFLReplaceChar	-
DFLReplaceString	-
DFLRightAlign	-
DFLRightAlignChar	-
DFLRightAlignNum	-
DFLSetLength	-
DFLStr	-
DFLStrCopy	-
DFLStrLower	Lcase(S)
DFLStrNSet	-
DFLStrSet	-
DFLStrToInt	-
DFLStrToIntDef	-
DFLStrUpper	Ucase(S): D

6.4.3.1 DFLAddQuote

Syntax

Visual Basic

```
DFLAddQuote(S as String) As String
```

Details

Return Value: String
Adds a quote at the end of a string.

Table 6-21: Parameters

Parameter	Description
S As String	A string

Example:

```
Dim RES As String
RES= DFLAddQuote("this is a test")
//the value of the variable RES is equal to «this is a test'»
```

6.4.3.2 DFLASCII

Return Value: Long
The ASCII code corresponding to the character S.

Syntax

Visual Basic

```
DFLASCII(SChar As String) As Long
```

Details

Table 6-22: Parameters

Parameter	Description
SChar As String	Character

Example:

```
DFLASCII("N") -> 78
```

6.4.3.3 DFLCharOccurenceCount

Return Value: Long
The number of occurrences of a character in a string.

Syntax

Visual Basic

```
DFLCharOccurenceCount(S As String, SChar As String) As Long
```

Details

Table 6-23: Parameters

Parameter	Description
S As String	String/body of text to search in
SChar As String	The character to look for

Example:

```
DFLCharOccurenceCount("character", "c") -> 2
```

6.4.3.4 DFLCheckFormat

Return Value: Boolean
True, if the format of the string argument *S* corresponds to the one indicated by the argument *Format*, False otherwise.

The argument *Format* can be a string, using the convention illustrated by the following example As 7N2A3X where N is numerical, A alphabetical (Latin alphabet), and X alphanumerical.

Syntax

Visual Basic

```
DFLCheckFormat(S As String, Format As String) As Boolean
```

Details

Table 6-24: Parameters

Parameter	Description
S As String	String to be tested
Format As String	String format

Example:

```
DFLCheckFormat("45VB","2N1A") -> False
DFLCheckFormat("45VB","2N2A") -> True
```

6.4.3.5 DFLCommaText

This function transforms the argument *Table* into a string argument *S* where all the elements of the table are surrounded by Double quotes, separated by commas and concatenated, or vice-versa, depending on the argument *TabToStr*.

Return Value: Boolean
True if it runs correctly, False otherwise.

Syntax

Visual Basic

`DFLCommaText(`*S* `As string,` *Tab* `As Variant,` *DimTab* `As Long,` *TabToStr* `As Boolean) As Boolean`

Details

Table 6-25: Parameters

Parameter	Description
S As String	A string in which the different elements are separated by Double quotes (CHR(34)) and commas. It is an Input or Output argument, depending on the value of the argument TabToStr, respectively False or True.
Tab As String(N)	A table/list with different elements. It is an Input or Output argument, depending on the value of the argument TablToStr, respectively True or False.
DimTab As Long	It indicates the dimension of the argument Table. It is an Input or Output argument, depending on the value of the argument TabToStr, respectively True or False.
TabToStr As Boolean	If True, the string S is filled from the argument Table. If False, it is the other way around.

Example:

```
Dim OutputTable(100) As String
Dim InputTable(100) As String
Dim OutputString As String, InputString As String, S As String
Dim NbElements As Long
" Transformation of the table towards to the string -- DimTable is then an input argument
InputTable(0)="First Element As a b c"
```

```
InputTable(1)="Second Element with quotes As
"+Chr(34)+"a"+Chr(34)+"b"+Chr(34)+"c"+Chr(34)
InputTable(2)="Third element with commas As "+Chr(44)+"a"+Chr(44)+"b"+Chr(44)+"c"+Chr(44)
InputTable(3)="Fourth element with quotes and commas As
"+Chr(34)+"a"+Chr(34)+Chr(44)+Chr(34)
    +"b"+Chr(34)+Chr(44)+Chr(34)+"c"+Chr(34)
If DFLCommaText(OutputString,InputTable,4,True) = True Then
MsgBox("The string resulting from the transformation of the table
is:"+Chr(10)+OutputString)
Else
MsgBox("Error CommaText")
End If
" Transformation of the string towards the table -- DimTable is then an output argument
InputString =
Chr(34)+"one"+Chr(34)+Chr(44)+Chr(34)+"two"+Chr(34)+Chr(44)+Chr(34)+"three"+Chr(34)
MsgBox("The input string is:"+Chr(10)+InputString)
If DFLCommaText(InputString,OutputTable,NbElements,False) = True Then
S = "The elements of the table resulting from the transformation of the string are:"
For i=0 To NbElements-1
S=S+Chr(10)+DFLIntToStr(i)+":"+OutputTable(i)
Next i
MsgBox(S)
Else
MsgBox("Error CommaText")
End If
```

6.4.3.6 DFLCommonCharCount

Return Value: Long

The number of identical characters at the same position in the two strings, indicated by the arguments *S1* and *S2*. If the two strings do not have the same length, the function returns -1.

Syntax

Visual Basic

`DFLCommonCharCount(`*S1* `As String,` *S2* `As String) As Long`

Details

Table 6-26: Parameters

Parameter	Description
S1 As String	First string
S2 As String	Second string

Example:

```
DFLCommonCharCount("character", "charictor") -> 7
Strings of different sizes:
DFLCommonCharCount("Test", "Testing") -> -1
```

6.4.3.7 DFLCommonNLeft

Return Value: Long
The number of identical characters starting from the beginning of two strings.

Syntax

Visual Basic

`DFLCommonNLeft(`*S1* `As String, `*S2* `As String) As Long`

Details

Table 6-27: Parameters

Parameter	Description
S1 As String	First string
S2 As String	Second string

Example:

```
DFLCommonNLeft("character", "charictor") -> 4
DFLCommonNLeft("character", "charactor") -> 7
```

6.4.3.8 DFLCommonNRight

Return Value: Long
The number of identical characters starting from the end of two strings.

Syntax

Visual Basic

`DFLCommonNRight(`*S1* `As String, `*S2* `As String) As Long`

Details

Table 6-28: Parameters

Parameter	Description
S1 As String	First string
S2 As String	Second string

Example:

```
DFLCommonNRight("character" , "charictor" ) -> 1
DFLCommonNRight("character" , "charicter" ) -> 4
```

6.4.3.9 DFLCompareStrings

Return Value: Long

The percentage of similarity between the two strings *S1* and *S2* with differences being weighed (subtracted from 100) as follows:

- Insertion weight As three
- Suppression weight As three
- Suppression weight for an unknown character As three
- Substitution weight As four
- Substitution weight for an unknown character As two

The closer the strings *S1* and *S2*, the higher the result returned by the function (two identical strings giving the number 100).

Syntax

Visual Basic

```
DFLCompareStrings(S1 As String, S2 As String) As Long
```

Details

Table 6-29: Parameters

Parameter	Description
S1 As String	String of characters, normally issued from the recognition phase.
S2 As String	Reference string of characters, possibly from a database.

Both parameters must not contain surrogate pairs.

Example:

```
DFLCompareStrings("Paris", "Peris") -> 80
```

6.4.3.10 DFLCompareStringsEx

Return Value: Long

The percentage of similarity between the two strings *S1* and *S2* with differences being weighed (subtracted from 100) as passed by the arguments.

The closer the strings *S1* and *S2*, the higher the result returned by the function (two identical strings giving the number 100).

Syntax

Visual Basic

`DFLCompareStringsEx(`*S1* `As String,` *S2* `As String,` *InsCost* `As Long,` *DelCost* `As Long,` *DelCostUC* `As Long,` *SubCost* `As Long,` *SubCostUC* `As Long) As Long`

Details

Table 6-30: Parameters

Parameter	Description
S1 As String	String of characters, normally issued from the recognition phase (must not contain surrogate pairs).
S2 As String	Reference string of characters, possibly from a database (must not contain surrogate pairs).
InsCost As Long	Insertion weight
DelCost As Long	Suppression weight
DelCostUC As Long	Suppression weight for an unknown character
SubCost As Long	Substitution weight
SubCostUC As Long	Substitution weight for an unknown character

Example:

```
DFLCompareStringsEx("Paris", "Peris", 1,1,1,1,2) -> 90
DFLCompareStringsEx("Paris", "Peris", 1,1,1,2,2) -> 80
```

6.4.3.11 DFLFloatToStr

Return Value: String
The string representation of a numerical value.

Syntax

Visual Basic

```
DFLFloatToStr(Num As Double) As String
```

Details

Table 6-31: Parameters

Parameter	Description
Num As Double	Numeric value

Example:

```
DFLFloatToStr(12.01) -> "12.01"
DFLFloatToStr(12.0) -> "12"
```

6.4.3.12 DFLFloatToStrF

Return Value: String
The string representation of a float D.

This function converts a float D into its string representation using the settings *Format*, *Precision*, and *DecimalCount*.

Syntax

Visual Basic

```
DFLFloatToStrF(D As Double, Format As String, Precision As Long,
DecimalCount As Long) As String
```

Details

Table 6-32: Parameters

Parameter	Description
D As Double	The float value.

Parameter	Description
Format As String	The output format (“GENERAL” or “EXPONENT” or “FIXED” or “NUMBER” or “CURRENCY”) **Note:** The output *Formats* depend on the global variables. For example, the *Format* CURRENCY is converted into a string representing a financial figure. The conversion is controlled by the global variables `CurrencyString`, `CurrencyFormat`, `NegCurrFormat`, `ThousandSeparator`, and `DecimalSeparator`, all of which are initialized from the **Regional Settings** of the configuration panel of Windows. To open this element of the configuration panel, select **Start** > **Settings** > **Configuration Panel**, then double-click on the **Regional Settings** icon.
Precision As Long	The precision to display D within the output string.
DecimalCount As Long	The maximum number of decimal digits to display D within the output string. **Note:** The number of digits following the decimal point is given by the argument `DecimalCount`, and must be between 0 and 18.

Table 6-33: Examples of `DFLFloatToStrF` function output values

Input value	Output value - English (United States) operating system	Output value - French (France) operating system
`DFLFloatToStrF(0.5, "EXPONENT", 8, 5)`	`"5.0000000E-1"`	`"5,0000000E-1"`
`DFLFloatToStrF(1234, "FIXED", 4, 2)`	`"1234.00"`	`"1234,00"`
`DFLFloatToStrF(1234, "NUMBER", 4, 2)`	`"1,234.00"`	`"1 234,00"`
`DFLFloatToStrF(1234, "CURRENCY", 4, 2)`	`"$1,234.00"`	`"1 234,00 €"`
`DFLFloatToStrF(1234, "GENERAL", 4, 2)`	`"1234"`	`"1234"`
`DFLFloatToStrF(-1234, "EXPONENT", 8, 3)`	`"-1.2340000E+003"`	`"-1,2340000E+003"`
`DFLFloatToStrF(-1234, "FIXED", 8, 3)`	`"-1234.000"`	`"-1234,000"`

Input value	Output value - English (United States) operating system	Output value - French (France) operating system
DFLFloatToStrF(-1234, "NUMBER", 8, 3)	"-1,234.000"	"-1 234,000"
DFLFloatToStrF(1234, "EXPONENT", 4, 2)	"1.234E+03"	"1,234E+03"
DFLFloatToStrF(-1234, "GENERAL", 8, 3)	"-1234"	"-1234"
DFLFloatToStrF(0, "GENERAL", 5, 2)	"0"	"0"
DFLFloatToStrF(0.5, "FIXED", 8, 5)	"0.50000"	"0,50000"
DFLFloatToStrF(0.5, "NUMBER", 8, 5)	"0.50000"	"0,50000"
DFLFloatToStrF(0.5, "CURRENCY", 8, 5)	"$0.50000"	"0,50000 €"
DFLFloatToStrF(0.5, "GENERAL", 8, 5)	"0.5"	"0,5"
DFLFloatToStrF(0, "EXPONENT", 5, 2)	"0.0000E+00"	"0,0000E+00"
DFLFloatToStrF(0, "FIXED", 5, 2)	"0.00"	"0,00"
DFLFloatToStrF(0, "NUMBER", 5, 2)	"0.00"	"0,00"
DFLFloatToStrF(0, "CURRENCY", 5, 2)	"$0.00"	"0,00 €"

6.4.3.13 DFLFormatAmount

Return Value: String

The amount formatted with the number of decimal digits specified in the parameters. If the output string has fewer decimal digits than the input string, then this function returns the nearest rounded value.

Enables an amount to be formatted with a number of decimal digits, as indicated in the argument `DecimalCount`.

Any character can be used to separate the decimals in the output string, using the argument `DecimalSeparator`.

Syntax

Visual Basic

```
DFLFormatAmount(Amount As String, DecimalCount As Long, DecimalSeparator
As String) As String
```

Details

Table 6-34: Parameters

Parameter	Description
Amount As String	String to be formatted
DecimalCount As Long	Number of decimal digits to be displayed in the output string
DecimalSeparator As String	Decimal separator to be used in the output string

Example:

```
DFLFormatAmount("24.5",2,".") -> "24.50"
```

6.4.3.14 DFLFormatVal

Return Value: String
The string representation of a numerical value *Num*.

This function converts a numerical value *Num*, into its string representation, using the argument *Format*.

The argument *Format* describes the manner in which the numerical value will be rendered.

It can be defined using the following characters:

- `0` Digit for a number. If the formatted value has a number at the position of the "0", then the number is copied in the output string. Otherwise, a "0" is included at the position.
- `#` Digit for a number. If the formatted value has a number at the position of the "#", then the number is copied in the output string. Otherwise, nothing is included at this position in the output string.
- `.` Decimal separator.
- `,` Thousands separator.
- `E+` Scientific Notation.
- `;` Separator for different formats for positive numbers, negative numbers, and zero.

Syntax

Visual Basic

```
DFLFormatVal(Format As String, Num As Double) As String
```

Details

Table 6-35: Parameters

Parameter	Description
Format As String	Format string expression
Num As Double	Numeric value to be formatted

Note: The format of the output value is controlled by the global variables `ThousandSeparator`, and `DecimalSeparator`, which are initialized from the **Regional Settings** of the configuration panel of Windows. To open this element of the configuration panel, select **Start** > **Settings** > **Configuration Panel**, then double-click on the **Regional Settings** icon.

The following examples show format strings and their action on four different values. In most cases, the empty *Format* string "" will answer most needs. Be aware that results are based on **Regional Settings** and separators, for example, can differ somewhat from the examples presented here.

Example:

```
DFLFormatVal("#,##0.00" ,1234) -> "1,234.00"
DFLFormatVal("0", 1234) -> "1234"
DFLFormatVal("#.###E-0", 1234) -> "1.234E3"
DFLFormatVal("0.00", 1234) -> "1234.00"
DFLFormatVal("0.000E+00", 1234) -> "1.234E+03"
DFLFormatVal("#.##", 1234) -> "1234"
DFLFormatVal("", 1234) -> "1234"
DFLFormatVal("#,##0.00;;Zero", 1234) -> "1,234.00"
DFLFormatVal("#,##0.00;(#,##0.00)", 1234) -> "1,234.00"
DFLFormatVal("", -1234) -> "-1234"
DFLFormatVal("0", -1234) -> "-1234"
DFLFormatVal("0.00", -1234) -> "-1234.00"
DFLFormatVal("#.##", -1234) -> "-1234"
DFLFormatVal("#,##0.00", -1234) -> "-1,234.00"
DFLFormatVal("0.000E+00", -1234) -> "-1.234E+03"
DFLFormatVal("#.###E-0", -1234) -> "-1.234E3"
DFLFormatVal("#,##0.00;;Zero", -1234) -> "-1,234.00"
DFLFormatVal("#,##0.00;(#,##0.00)" , -1234) -> "(1,234.00)"
DFLFormatVal("#.###E-0" , 0.5) -> "5E-1"
DFLFormatVal("", 0.5) -> "0.5""
DFLFormatVal("0" , 0.5) -> "1"
DFLFormatVal("0.000E+00", 0.5) -> "5.000E-01"
DFLFormatVal("#,##0.00", 0.5) -> "0.50"
DFLFormatVal("0.00", 0.5) -> "0.50"
DFLFormatVal("#,##0.00;;Zero", 0.5) -> "0.50"
DFLFormatVal("#.##", 0.5) -> ".5"
DFLFormatVal("#,##0.00;(#,##0.00)", 0.5) -> "0.50"
DFLFormatVal("#,##0.00;;Zero", 0) -> "Zero"
DFLFormatVal("#.##", 0) -> ""
DFLFormatVal("0", 0) -> "0"
DFLFormatVal("#,##0.00;(#,##0.00)", 0) -> "0.00"
DFLFormatVal("0.000E+00", 0) -> "0.000E+00"
DFLFormatVal("#.###E-0", 0) -> "0E0"
DFLFormatVal("", 0) -> "0"
DFLFormatVal("#,##0.00", 0) -> "0.00"
DFLFormatVal("0.00", 0) -> "0.00"
Dim x As Long
x=456
DFLFormatVal("",x) -> "456"
Dim x As Long
```

```
x=144.5
DFLFormatVal("",x) -> "144.5"
Dim x As Double
x=144.52
DFLFormatVal("",x) -> "144.520004272461"
```

This last case shows the problems of rounding linked to the computer coding of floats. It shows the interest of the Format string to limit the number of digits after the decimal separator, and eventually reformat the Integer part with 0.

Example: Limiting the Number of digits After the Decimal Separator

```
Dim x As Double
x=144.52
DFLFormatVal("0.0000",x) -> "144.5200"
DFLFormatVal("0.0",x) -> "144.5"
```

The code "0" allows you to specify the number of significant digits to show after the decimal separator.

Example: Used for the Integer Part, It Formats It With Eventual 0:

```
Dim x As Double
x=144.52
DFLFormatVal("00000.00",x) -> "00144.52"
```

6.4.3.15 DFLGetLength

Return Value: Long

The number of characters present in a string *S*.

Syntax

Visual Basic

`DFLGetLength(`*S* `As String) As Long`

Details

Table 6-36: Parameters

Parameter	Description
S As String	String

Note: For characters encoded by a surrogate pair, the function counts two characters and then returns two and not one.

Example:

```
DFLGetLength("this is a test") -> 16
```

```
DFLGetLength("") -> 0
```

6.4.3.16 DFLIntToStr

Return Value: String
The string representation of an integer value.

Syntax

Visual Basic

```
DFLIntToStr(Num As Long) As String
```

Details

Table 6-37: Parameters

Parameter	Description
Num As Long	Numeric value

Example:

```
DFLIntToStr(3) -> "3"
```

6.4.3.17 DFLIsAmount

Return Value: Boolean
True if a string of characters is an amount type. False otherwise.

Verifies that a string of characters is an amount type, with a specified number of decimal digits.

Syntax

Visual Basic

```
DFLIsAmount(Amount As String, DecimalCount As Long) As Boolean
```

Details

Table 6-38: Parameters

Parameter	Description
Amount As String	String to be checked.
DecimalCount As Long	Number of decimal digits expected.

Example:

```
DFLIsAmount("24,50",2) -> True
```

```
DFLIsAmount("24.50",2) -> True
```

```
DFLIsAmount("24,5",2) -> False
DFLIsAmount("24.5",2) -> False
```

Note: The decimal separator can be the point (full stop) or the comma, independently of regional settings in Windows Control Panel.

6.4.3.18 DFLIsInteger

Return Value: Boolean

True if the argument *S* is a positive integer. False otherwise. The empty string also returns True.

Syntax

Visual Basic

```
DFLIsInteger(S As String) As Boolean
```

Details

Table 6-39: Parameters

Parameter	Description
S As String	String to be tested.

Example:

```
DFLIsInteger("1234") -> True
DFLIsInteger("") -> True
DFLIsInteger("This is a test") -> False
```

6.4.3.19 DFLKeepChar

Enables filtering of field values, preserving only some characters.

Return Value: String

Filtered output value with the filtered string and the number of characters taken out, separated by a comma.

Syntax

Visual Basic

```
DFLKeepChar(FieldValueIn As String, KeepNum As String, KeepAlpha As
string, Other As String) As String
```

Details

Table 6-40: Parameters

Parameter	Description
FieldValueIn As String	Field value as input.
KeepNum As String	"0" or "1", depending if you want to preserve numerical characters (1) or no (0)
KeepAlpha As String	"0", "1" , "2" or "3", depending if you want to preserve uppercase alphabetical characters (1), uppercase and lowercase alphabetical characters (2), lowercase alphabetical characters (3), or neither (0). **Note:** Does not apply to Asian languages because characters are neither uppercase nor lowercase.
Other As String	List of the other characters to preserve, as a string. For example: ".,+-*/文件"

Example:

```
DFLKeepChar("This is the test number 1","1","1","+") -> "T1, 23"
```

6.4.3.20 DFLLeftAlign

Return Value: String

The string argument S padded with spaces to the right so as to output a string, the length of which is determined by the argument *NewLength*. The original string argument *S* will be aligned to the left of the output string.

Syntax

Visual Basic

```
DFLLeftAlign(S As String, NewLength As Long) As String
```

Details

Table 6-41: Parameters

Parameter	Description
S As String	String to be formatted.
NewLength As Long	Length of returned string

Example:

```
DFLLeftAlign("PARIS" , 3) -> "PAR"
DFLLeftAlign("PARIS" , 8) -> "PARIS     "
```

6.4.3.21 DFLLeftAlignChar

Return Value: String

The string argument *S* padded with the character of the argument *SChar* to the right so as to output a string, the length of which is determined by the argument *NewLength*. The original string argument *S* will be aligned to the left of the output string.

Syntax

Visual Basic

```
DFLLeftAlignChar(S As String, SChar As String, NewLength As Long) As
String
```

Details

Table 6-42: Parameters

Parameter	Description
S As String	String to be formatted.
SChar As String	Character to add.
NewLength As Long	Length of returned string

Example:

```
DFLLeftAlignChar("123", "c", 5) -> "123cc"
```

6.4.3.22 DFLLeftAlignNum

Return Value: String

The string argument *S* padded with 0 (zeros) to the right so as to output a string, the length of which is determined by the argument *NewLength*. The original string argument *S* will be aligned to the left of the output string.

Syntax

Visual Basic

```
DFLLeftAlignNum(StringToBeFormatted As String, NewLength As Long) As
String
```

Details

Table 6-43: Parameters

Parameter	Description
S As String	String to be formatted.
NewLength As Long	Length of returned string

Example:

```
DFLLeftAlignNum( "123" , 5 ) -> "12300"
```

6.4.3.23 DFLPos

Return Value: Long

The position of the first occurrence of the string indicated by the argument *SubStr* in the string argument S. If the string argument *SubStr* is not found in the string argument *S*, this function returns 0.

Syntax

Visual Basic

```
DFLPos(SubStr As String, S As String) As Long
```

Details

Table 6-44: Parameters

Parameter	Description
SubStr As String	Substring to find
S As String	String

Both parameters must not contain surrogate pairs.

Example:

```
DFLPos( "r" , "character" ) -> 4
```

```
DFLPos( "ac" , "character" ) -> 5
```

6.4.3.24 DFLRegExp

Searches for matches of a defined pattern using a "Regular Expression" syntax in a string of characters. A regular expression is a string used to describe or match a set of characters, according to certain syntax rules. During recognition step, regular expressions can, for example, search for specific characters, the position of characters in a string, or specific grouping of characters. Refer to the topic *Understanding regular expressions* in the *Intelligent Capture Designer Guide* for more information on using and creating regular expressions.

Return Value: Boolean

True if the operation was completed successfully, False otherwise. If False, the argument ErrMsg then includes the explanation of the error.

Syntax

Visual Basic

```
DFLRegExp(ExprToFind As String, S As String, NbMax As Long, PosTab As
Variant, LenTab As Variant, ErrMsg As String) As Boolean
```

Details

Table 6-45: Parameters

Parameter	Description
ExprToFind As String	Pattern to look for, using the syntax of "Regular Expressions".
S As String	String to search the pattern in (must not contain surrogate pairs).
NbMax As Long	Maximum number of returns. This parameter is then modified to indicate the exact number of returns (less than or equal to its initial input value).
PosTab As Long(N)	List of positions where the pattern has been found.
LenTab As Long(N)	List of lengths of the found patterns.
ErrMsg As String	Specify the problem encountered.

Example:

```
integer Dim iNbrMax As Long
integer Dim iTabPos([4)] As Long
integer Dim iTabLen([4)] As Long
string Dim stErrMsg As String
iNbrMax = 4
OK = DFLRegExp('"[0-9]{5}|SW'",'"SWT, 21 rue des genets, 94310 Orly Ville'", iNbrMax,
iTabPos, iTabLen, stErrMsg)
// iNbrMax contains the value 2 corresponding to the effective number of occurrences
found.
```

```
// iTabPos[(0)] contains the value 1, iTabLen([0)] contains the value 2
// corresponding to '"SW'" being found in the address string
// iTabPos([1)] contains the value 25, iTabLen([1)] contains the value 5
// corresponding to '"94310'" being found in the address string
```

Note: The parameter *NbMax* is both an input and output parameter, and its meaning as output is different from its meaning as input.

The parameter *PosTab* and *Len Tab* are Variant but must be declared as tables of long otherwise there is an error message when calling DFLRegExp.

6.4.3.25 DFLReplaceChar

Return Value: String

The string made of the string argument *AText* with all the occurrences of the character *CharFromText* replaced by the character *CharToText*. This function is not case sensitive, and therefore all occurrences are replaced.

Syntax

Visual Basic

DFLReplaceChar(*AText* As String, *CharFromText* As String, *CharToText* As String) As String

Details

Table 6-46: Parameters

Parameter	Description
AText As String	String to formatted
CharFromText As String	Character to be replaced from AText
CharToText As String	Character To replace CharFromText

Example:

```
DFLReplaceChar("%string%to%modify%", "%", "_") -> "_string_to_modify_"
```

6.4.3.26 DFLReplaceString

Return Value: String

The string made of the string argument *AText* with all the occurrences of the substring *StrFromText* replaced by the substring *StrToText*. This function is not case sensitive, and therefore all occurrences are replaced.

Syntax

Visual Basic

```
DFLReplaceString(ATextAs String, StrFromText As String, StrToText As String) As String
```

Details

Table 6-47: Parameters

Parameter	Description
AText As String	String to formatted
StrFromText As String	String to be replaced from AText
StrToText As String	String to replace StrFromText

Example:

```
DFLReplaceString( "This is a test. Test this function and see how it can take care of multiple occurrences" , "Test" , "new Word" ) -> "This is a new Word. new Word this function and see how it can take care of multiple occurrences"
```

6.4.3.27 DFLRightAlign

Return Value: String

The string argument *S* padded with spaces to the left so as to output a string, the length of which is determined by the argument *NewLength*. The original string argument *S* is aligned to the right of the output string.

Syntax

Visual Basic

```
DFLRightAlign(S As String, NewLength As Long) As String
```

Details

Table 6-48: Parameters

Parameter	Description
S As String	String to be formatted.
NewLength As *Long*	Length of returned string

Example:

```
DFLRightAlign("PARIS", 8) -> "   PARIS"
```

6.4.3.28 DFLRightAlignChar

Return Value: String

The string argument *S* padded with the character of the argument *SChar* to the left so as to output a string, the length of which is determined by the argument *NewLength*. The original string argument *S* will be aligned to the right of the output string.

Syntax

Visual Basic

```
DFLRightAlignChar(StringToBeFormatted As String, Car As String,
NewLength As Long) As String
```

Details

Table 6-49: Parameters

Parameter	Description
S As String	String to be formatted.
SChar As String	Character to add.
NewLength As Long	Length of returned string

Example:

```
DFLRightAlignChar("123", "c", 5) -> "cc123"
```

6.4.3.29 DFLRightAlignNum

Return Value: String

The string argument *S* padded with 0 (zeros) to the left so as to output a string, the length of which is determined by the argument *NewLength*. The original string argument *S* will be aligned to the right of the output string.

Syntax

Visual Basic

```
DFLRightAlignNum(S As String, NewLength As Long) As String
```

Details

Table 6-50: Parameters

Parameter	Description
S As String	String to be formatted.
NewLength As Long	Length of returned string

Example:

```
DFLRightAlignNum("123", 5) -> "00123"
```

6.4.3.30 DFLSetLength

Return Value: String

The new string with the length specified in the parameters.

Used to modify the length of a string of characters.

If the length is greater than the initial length S, S will be filled in with spaces.

Syntax

Visual Basic

```
DFLSetLength(S As String, Len As Long) As String
```

Details

Table 6-51: Parameters

Parameter	Description
S As String	String to be formatted (must not contain surrogate pairs).

Parameter	Description
Len As Long	Length of returned string (number of characters)

Example:

```
DFLSetLength("PARIS", 8) -> "PARIS   "
```

6.4.3.31 DFLStr

Return Value: String

A String converted from a Double, with three decimal digits and a dot as a separator.

Syntax

Visual Basic

```
DFLStr(Num As Double) As String
```

Details

Table 6-52: Parameters

Parameter	Description
Num As Double	Numeric value

Example: *DFLStr(12.0) -> "12.000"*

DFLStr(12.01) -> "12.010"

6.4.3.32 DFLStrCopy

Return Value: String

A substring (the size of which is given by the argument *Size*) of the string S, from the position indicated by the argument *Index*. The first character in the string *S* is in position one.

If the number of characters to be extracted goes beyond the end of the string S, only the characters between S[Index] and the end of S are returned.

Syntax

Visual Basic

```
DFLStrCopy(S As String, Index As Long, Size As Long) As String
```

Details

Table 6-53: Parameters

Parameter	Description
S As String	String to extract the substring from (must not contain surrogate pairs)
Index As Long	Position to extract the substring from
Count As Long	Size of the substring to extract

Example:

```
DFLStrCopy( "this is a test" , 1 , 4 ) -> "this"
```

6.4.3.33 DFLStrLower

Return Value: String

String of characters converted to lowercase.

Note: Does not apply to Asian languages because characters are neither uppercase nor lowercase.

Syntax

Visual Basic

```
DFLStrLower(S As String) As String
```

Details

Table 6-54: Parameters

Parameter	Description
S As String	String to convert

Example:

```
DFLStrLower( "This IS A TEST" ) -> "this is a test"
```

6.4.3.34 DFLStrNSet

Return Value: String

A string made from *S* with the first *Count* characters of the string *SubStr* overwriting characters from the position *Index,* increasing the length of the output string if necessary.

Syntax

Visual Basic

```
DFLStrNSet(S As String, Index As Long, Count As Long, SubStr As String) As String
```

Details

Table 6-55: Parameters

Parameter	Description
S As String	Original string (must not contain surrogate pairs)
Index As Long	Position to replace from
Count As Long	Size of the substring to insert
SubStr As String	Substring to insert (must not contain surrogate pairs)

Example:

```
DFLStrNset("this is a test", 11, 8, "new test") -> "this is a new test"
```

6.4.3.35 DFLStrSet

Return Value: String

A string made from *S* with the string *SubStr* overwriting characters from the position *Index,* increasing the length of the output string if necessary.

Syntax

Visual Basic

```
DFLStrSet(S As String, Index As Long, SubStr As String) As String
```

Details

Table 6-56: Parameters

Parameter	Description
S1 As String	Original String (must not contain surrogate pairs)
Index As Long	Position to replace from
SubStr As String	Substring to replace with (must not contain surrogate pairs)

Example:

```
DFLStrSet("Hello", 3, "This is a test") -> "HeThis is a test"
DFLStrSet("this is a test", 6, "xxx") -> "this xxxa test"
```

6.4.3.36 DFLStrToInt

Return Value: Long

An integer converted from the string *S*.

In the case where the input string is not interpretable as a string or if it is empty, the use of the function `StrToInt` will cause an error at run time.

Syntax

Visual Basic

```
DFLStrToInt(S As String) As Long
```

Details

Table 6-57: Parameters

Parameter	Description
S As String	String

Example:

```
DFLStrToInt("4") -> 4
```

6.4.3.37 DFLStrToIntDef

Converts a string *S* into an integer.

Return Value: Long
If the string S does not correspond to any recognized number, `DFLStrToIntDef` returns the number of the argument Default.

Syntax

Visual Basic

```
DFLStrToIntDef(S As String, DefaultValue As Long) As Long
```

Details

Table 6-58: Parameters

Parameter	Description
S As String	String
Default As Long	Default integer value

Example:

```
DFLStrToIntDef( "4" , 10 ) -> 4
DFLStrToIntDef( "az4" , 10 ) -> 10
```

6.4.3.38 DFLStrUpper

Return Value: String
String of characters converted to uppercase.

Note: Does not apply to Asian languages because characters are neither uppercase nor lowercase.

Syntax

Visual Basic

```
DFLStrUpper(S As String) As String
```

Details

Table 6-59: Parameters

Parameter	Description
S As String	String

Example:

```
DFLStrUpper( "This is a test" ) -> "THIS IS A TEST"
```

6.4.4 DFL Date functions

Where possible, *VBA* functions should be used instead of Dispatcher library functions.

Table 6-60: Dispatcher Date Functions and *VBA* Equivalents

Function	VBA equivalent
DFLCheckDate	-
DFLCheckDateDDMMYYYY	-
DFLCompareDates	-
DFLDateTostr	CDate(Dt) : S
DFLDayOfYear	-
DFLDaysBetweenDates	-
DFLDecodeDate	-
DFLEncodeDate	-
DFLGetDate	-
DFLStrToDate	-

6.4.4.1 DFLCheckDate

Return Value: Boolean

True, if the date indicated by the arguments Day/Month/Year is valid. False, otherwise.

Syntax

Visual Basic

```
DFLCheckDate(LDay As Long, LMonth As Long, LYear As Long) As Boolean
```

Details

Table 6-61: Parameters

Parameter	Description
LDay As Long	Day value expressed in two digits DD
LMonth As Long	Month value expressed in two digits MM
LYear As Long	Year value expressed in four digits YYYY

Example:

```
DFLCheckDate(36, 1, 1999) -> False
DFLCheckDate(1, 1, 1999) -> True
```

6.4.4.2 DFLCheckDateDDMMYYYY

Checks that the argument *SDate* corresponds to the format DDMMYYYY

Return Value: Boolean

True if the string corresponds to the date format. False, otherwise.

Syntax

Visual Basic

```
DFLCheckDateDDMMYYYY(SDate As String) As Boolean
```

Details

Table 6-62: Parameters

Parameter	Description
SDate As String	String to check against the date format DDMMYYYY

Example:

```
OK = DFLCheckDateDDMMYYYY("25122003")
//then the value of the variable OK is equal to 1 (true)
```

6.4.4.3 DFLCompareDates

Return Value: Long

- 0 if the two dates are identical
- 1 if the starting date > ending date
- -1 if the starting date < ending date
- 2 if the function encountered a problem

Syntax

Visual Basic

```
DFLCompareDates(FirstDay As Long, FirstMonth As Long, FirstYear As Long,
LastDay As Long, LastMonth As Long, LastYear As Long) As Long
```

Details

Table 6-63: Parameters

Parameter	Description
FirstDay As Long	Day value expressed in two digits (DD)
FirstMonth As Long	Month value expressed in two digits (MM)
FirstYear As Long	Year value expressed in four digits (YYYY)
LastDay As Long	Day value expressed in two digits (DD)
LastMonth As Long	Month value expressed in two digits (MM)
LastYear As Long	Year value expressed in four digits (YYYY)

Example:

```
DFLCompareDates(1, 1, 2003, 1, 1, 2003) -> 0
DFLCompareDates(10, 1, 2003, 1, 1, 2003) -> 1
DFLCompareDates(1, 1, 2003, 10, 1, 2003) -> -1
DFLCompareDates(36, 1, 2003, 10, 1, 2003) -> 2
```

6.4.4.4 DFLDateTostr

Return Value: String
The date in the form of a string of characters.

Syntax

Visual Basic

```
DFLDateToStr(DDate As Date) As String
```

Details

Table 6-64: Parameters

Parameter	Description
DDate As Date	Date value

Note: The date format is controlled by the **Regional Settings** in the configuration panel of Windows. To open this element of the configuration panel, select **Start** > **Settings** > **Configuration Panel**, then double-click on the **Regional Settings** icon. Be aware that since results are based on **Regional Settings** and can differ somewhat from the example presented here.

Example:

```
DFLDateToStr(Now) -> "17/07/2007" // "Now" variable gives the current day
DFLDateToStr(40000) -> "06/07/2009"
```

6.4.4.5 DFLDayOfYear

Return Value: String

The number of the current day within the current year. For example if the current day is Nov. 17 2008, the function returns 322.

Syntax

Visual Basic

```
DFLDayOfYear() As String
```

Details

PARAMETERS

None

Example:

```
Dim stDayOfYear As String
stDayOfYear = DFLDayOfYear()
```

6.4.4.6 DFLDaysBetweenDates

Return Value: Long

The number of days between the starting date and the ending date, both dates being included.

Syntax

Visual Basic

```
DFLDaysBetweenDates(FirstDay As Long, FirstMonth As Long, FirstYear As
Long, LastDay As Long, LastMonth As Long, LastYear As Long) As Long
```

Details

Table 6-65: Parameters

Parameter	Description
FirstDay As Long	Day value expressed in two digits (DD)
FirstMonth As Long	Month value expressed in two digits (MM)
FirstYear As Long	Year value expressed in four digits (YYYY)

Parameter	Description
LastDay As Long	Day value expressed in two digits (DD)
LastMonth As Long	Month value expressed in two digits (MM)
LastYear As Long	Year value expressed in four digits (YYYY)

Example:

```
DFLDaysBetweenDates(1, 1, 2003, 10, 1, 2003) -> 10
```

```
DFLDaysBetweenDates(1, 1, 2003, 1, 1, 2003) -> 1
```

6.4.4.7 DFLDecodeDate

Return Value: Long
The returned value is always one.

Sets the Year, the Month and the Day from the Date.

Syntax

Visual Basic

```
DFLDecodeDate (DDate As Date, LYear As Long, LMonth As Long, LDay As Long)
As Long
```

Details

Table 6-66: Parameters

Parameter	Description	Modified
DDate As Long	Day value	
LYear As Long	Year value expressed in four digits (YYYY)	Yes
LMonth As Long	Month value expressed in two digits (MM)	Yes
LDay As Long	Day value expressed in two digits (DD)	Yes

Example:

```
Dim y As Long
Dim m As Long
Dim d As Long
DFLDecodeDate(Now,y,m,d) -> y=2007, m=07, d=17
// "Now" variable gives the current day
```

6.4.4.8 DFLEncodeDate

Return Value: Date

The Date from parameter year *LYear*, month *LMonth* and day *LDay*.

Syntax

Visual Basic

```
DFLEncodeDate(LYear As Long, LMonth As Long, LDay As Long) As Date
```

Details

Table 6-67: Parameters

Parameter	Description
LYear As Long	Year value expressed in four digits (YYYY)
LMonth As Long	Month value expressed in two digits (MM)
LDay As Long	Day value expressed in two digits (DD)

Example:

```
Dim CurrDate As Date
CurrDate = DFLEncodeDate(2007,7,17)
Msgbox (Str(CurrDate))    -> // shows "7/17/2007"
Msgbox (DFLIntToStr(CurrDate)) -> // shows "39280"
```

6.4.4.9 DFLGetDate

Return Value: String

Today's date using the format as indicated by the argument *DateFormat*, using the following symbols:

- dd: 2 digit day
- mm: 2 digit month
- yyyy : 4 digit year
- hh: 2 digit hour
- nn: 2 digit minutes
- ss: 2 digit seconds

Syntax

Visual Basic

```
DFLGetDate(DateFormat As String) As String
```

Details

Table 6-68: Parameters

Parameter	Description
DateFormat As String	DateFormat

Example:

```
DFLGetDate("hhnnss") -> "150145"
DFLGetDate("ddmmyyyy") -> "17072007"
```

6.4.4.10 DFLStrToDate

Return Value: Date

The date provided in the string of characters *S*.

Syntax

Visual Basic

```
DFLStrToDate(SDate As String) As Date
```

Details

Table 6-69: Parameters

Parameter	Description
SDate As String	Date

Note: The date format in the input value must respect the format defined in the **Regional Settings** of the configuration panel of Windows. To open this element of the configuration panel, select **Start** > **Settings** > **Configuration Panel**, then double-click the **Regional Settings** icon. Be aware that results are based on **Regional Settings** and can differ somewhat from the example presented here.

Example:

```
Dim CurrDate As Date
CurrDate = DFLStrToDate("17/07/2007")
Msgbox (Str(CurrDate))    -> // shows "7/17/2007"
Msgbox (DFLIntToStr(CurrDate)) -> // shows "39280"
```

6.4.5 DFL Graphical Interface functions

Use the equivalent *VBA* function instead of a Dispatcher library function whenever possible.

Table 6-70: Graphical Interface Library and VBA Equivalents

Function	VBA equivalent
DFLDataGridInput	-
DFLInputQuery	-
DFLMessageBox	-
DFLMessageDlg	-
DFLMessageSetPosition	-
DFLShowData1	-
DFLShowData1FromFile	-
DFLShowData2	-
DFLShowData3	-
DFLShowData4	-
DFLShowDataN	-
DFLShowMessage	-

6.4.5.1 DFLDataGridInput

Displays a datasheet that enables the user to input data. The name of the columns are indicated by the argument *ColsName*. The **TAB**, **ENTER** and **ARROW** keys enable the user to move in the datasheet. At the end of a row, pressing **ENTER** adds a new row.

The user validates input by selecting **OK** or pressing **CTRL + ENTER**. The values entered in the cells are then archived in the argument *Cells* (filled from left to right and from top to bottom).

If *Cells* is larger than the number of displayed cells, the rest of the elements of *Cells* are empty strings. If *Cells* is too small to hold all of the typed in data, an error message is generated through the output value set to false.

The number of displayed cells *CellsCount* must be less or equal to the number of elements in the table argument *Cells*, which must be foreseen to be large enough.

Return Value: Boolean

True, if the operator confirmed the input values and if *Cells* size is sufficient, False, if the operator cancelled the input operation or if *Cells* size is not sufficient.

Syntax

Visual Basic

```
DFLDataGridInput(ColsName As Variant, ColsCount As Long, Cells As
Variant, CellsCount As Long) As Boolean
```

Details

Table 6-71: Parameters

Parameter	Description
ColsName(N) As String	Table of strings, detailing the title of the columns in the datasheet
ColsCount As Long	Number of columns to display
Cells(N) As String	Table of strings containing the values entered by operator
CellsCount As Long	Number of displayed cells

Table 6-72: Shortcut keys

Button	Shortcut key	Function
OK	**CTRL + ENTER**	Validate the input of the datasheet and archive the data in the argument Cells. You must foresee a sufficient capacity in the table Cells. The function returns True.
Add	**CTRL + ARROW DOWN**	Add a row in the input datasheet.
Remove	**CTRL + ARROW UP**	Delete the last row in the datasheet until there is only one left. If you then add the lines that were deleted, the data in the cells are restored.
Empty Cell	**CTRL + DELETE** (even if the text is not selected)	Erase the data contained in the cell currently being edited.
Empty Line	**CTRL + SPACE**	Erase the data contained in the row currently being edited.
Empty All	**F2**	Erase the data contained in all the cells and place the cursor on the top left cell.

Button	Shortcut key	Function
Cancel	**ESC**	Exit the input without archiving the data. The function returns False and the argument CellList is initialized with empty strings and CellCount is equal to ColumnCount.

Example:

```
Dim stInputTable(5) As String
Dim stOutputTable(10) As String
Dim iInputColumnCount As Long, iOutputCellCount As Long
Dim OK As Boolean
' Code for DFLDataGridInput
iInputColumnCount = 4
stInputTable(0)="Lodgement Date"
stInputTable(1)="Lodgement Reference Nb"
stInputTable(2)="Bank Code"
stInputTable(3)="Clerk"
OK=DFLDataGridInput(stInputTable,iInputColumnCount,stOutputTable,iOutputCellCount)
For i=0 To iOutputCellCount-1
' your code with stOutputTable[i]
Next i
```

Note: The parameter *ColsName* and *Cells* are Variant but must be declared as tables of String otherwise there is an error message when calling DFLDataGridInput.

6.4.5.2 DFLInputQuery

Return Value: Boolean
True if the user clicks **OK**. Otherwise, the function returns False.

Displays a window with the argument *Legend* as the title and the argument *Mess* as the message/question, and a control enabling the user to input a string.

If the function returns False, the argument *Value* is not changed.

Syntax

Visual Basic

```
DFLInputQuery(Legend As String, Mess As String, Value As String) As
Boolean
```

Details

Table 6-73: Parameters

Parameter	Description
Legend As String	Title
Mess As Long	Message/Question
Value As String	In/out value that can be modified by the user in the window

Example:

```
Dim EnteredValue As String
OK = DFLInputQuery("Value","Enter a value",EnteredValue)
```

6.4.5.3 DFLMessageBox

Return Value: Long

The number corresponding to the button the user pressed, starting with 0 if the user pressed the first button, 1 for the second, etc.

Displays the message passed in the argument *Mess* in a window with all the buttons contained in the *Buttons* list argument. This window disappears when the user selects either of these buttons.

The *Buttons* argument must be a list of the buttons to be displayed, separated by commas (for example: 'button1,button2,button3').

Syntax

Visual Basic

DFLMessageBox(*Mess* As String, *Buttons* As String) As Long

Details

Table 6-74: Parameters

Parameter	Description
Mess As String	Message/Question
Buttons As String	Buttons names

Example:

```
OK = DFLMessageBox("Do you want to display this message?", "Yes, No")
```

6.4.5.4 DFLMessageDlg

Return Value: Boolean
True if the user selected **Yes**. False, otherwise.

Displays the message passed in the argument *Message* in a window with two buttons **Yes** and **No**. This window disappears when the user selects either of these buttons.

Syntax

Visual Basic

```
DFLMessageDlg(Mess As String) As Boolean
```

Details

Table 6-75: Parameters

Parameter	Description
Mess As String	Message/Question

Example:

```
OK = DFLMessageDlg("Do you want to display this message?")
```

6.4.5.5 DFLMessageSetPosition

Enables positioning of the window using alignment options HAlign and VAlign.

- HAlign has to be one of the following options: “LEFT”, “CENTER” or “RIGHT”
- VAlign has to be one of the following options: “TOP” “CENTER” or “BOTTOM”

All the windows inherit the positions defined by `DFLMessageSetPosition`. By default, the window appears in the center of the screen.

Return Value: Boolean
True if the operation was completed successfully. False otherwise

Syntax

Visual Basic

```
DFLMessageSetPosition(HAlign As String, VAlign As String) As Boolean
```

Details

Table 6-76: Parameters

Parameter	Description
HAlign As String	Horizontal alignment
VAlign As String	Vertical alignment

Example:

```
DFLMessageSetPosition("CENTER", "BOTTOM") //Displays all the windows centered at the
bottom of the screen.
```

6.4.5.6 DFLShowData1

Displays a window with a selection list populated from the *Propositions1* argument.

Return Value: Long
The index of the selected line, or -1 if the user used the **ESC** key

Syntax

Visual Basic

DFLShowData1(*PropCountMax* As String, *ColTitle1* As String, *Propositions1* As Variant) As Long

Details

Table 6-77: Parameters

Parameter	Description
PropCountMax As Integer	The maximum number of propositions displayed on the screen
ColTitle1 As String	The title of the option list
Propositions1(N) As String	The list of values of the option list

Example:

```
Dim NameList(6) As String
NameList(0) = "McClain"
NameList(1) = "Kennedy"
NameList(2) = "Smith"
NameList(3) = "Nicols"
NameList(4) = "Jordan"
NameList(5) = "Bryant"
LineNumber = DFLShowData1(6, "Name", NameList)
```

Note: The *Propositions1* parameter is Variant but must be declared as a String, otherwise, an error message appears when calling `DFLShowData1`.

6.4.5.7 DFLShowData1FromFile

Displays a window with a list of propositions, which is filled from a file indicated by the *FullFileName* argument.

Return Value: String

The field value of the selected line, or an empty string if the user used the **ESC** key.

Syntax

Visual Basic

```
DFLShowData1FromFile(FullFileName As String, ColTitle As String) As String
```

Details

Table 6-78: Parameters

Parameter	Description
FullFileName As String	Full file name which contains the list of propositions
ColTitle As String	The title of the option list

Example:

```
LineValue = DFLShowData1FromFile("C:\Data\FileName.txt", "Name")
```

6.4.5.8 DFLShowData2

Displays a window with a list of propositions, which is filled from the *Propositions1* to 2 arguments.

Return Value: Long

The index of the selected line, or -1 if the user used the **ESC** key

Syntax

Visual Basic

```
DFLShowData2(PropCountMax As String, ColTitle1 As String, Propositions1 As Variant, ColTitle2 As String, Propositions2 As Variant) As Long
```

Details

Table 6-79: Parameters

Parameter	Description
PropCountMax As Integer	The maximum number of propositions displayed on the screen
ColTitle1 As String	The title of the propositions list
Propositions1(N) As String	The list of values of the propositions list
ColTitle2 As String	The title of the second column of the propositions list
Propositions2(N) As String	The list of values of the second column of the propositions list

Example:

```
Dim NameList(5) As String
Dim FirstNameList(5) As String
NameList(0) = "McClain
NameList(1) = "Kennedy"
NameList(2) = "Smith"
NameList(3) = "Nicols"
NameList(4) = "Jordan"
NameList(5) = "Bryant"
FirstNameList(0) = "John"
FirstNameList(1) = "Jeffrey"
FirstNameList(2) = "Anna"
FirstNameList(3) = "Monica"
FirstNameList(4) = "Lisa"
FirstNameList(5) = "Steve"
LineNumber = DFLShowData2(6, "NAME", NameList, "FIRSTNAME", FirstNameList)
```

Note: The *Propositions1* and *Propositions2* parameters are Variant but must be declared as tables of String otherwise there is an error message when calling `DFLShowData2`.

6.4.5.9 DFLShowData3

Displays a window with a list of propositions , which is filled from the *Propositions1*, *Propositions2* and *Propositions3* arguments.

The *Propositions1*, *Propositions2* and *Propositions3* parameters are Variant but must be declared as String tables, otherwise, an error message appears when calling `DFLShowData3`.

Return Value: Long

The index of the selected line, or -1 if the user used the **ESC** key

Syntax

Visual Basic

DFLShowData3(PropCountMax As String, *ColTitle1* As String, *Propositions1* As Variant, *ColTitle2* As String, *Propositions2* As Variant, *ColTitle3* As String, *Propositions3* As Variant) As Long

Details

Table 6-80: Parameters

Parameter	Description
PropCountMax As Integer	The maximum number of propositions displayed on the screen
ColTitle1 As String	The title of the propositions list
*Propositions1(N)*As String	The list of values of the propositions list
ColTitle2 As String	The title of the second column of the propositions list
Propositions2(N) As String	The list of values of the second column of the propositions list
ColTitle3 As String	The title of the third column of the propositions list
Propositions3(N) As String	The list of values of the third column of the propositions list

Example:

```
Dim NameList(5) As String
Dim FirstNameList(5) As String
Dim AgeList(5) As String
NameList(0) = "McClain"
NameList(1) = "Kennedy"
NameList(2) = "Smith"
NameList(3) = "Nicols"
NameList(4) = "Jordan"
NameList(5) = "Bryant"
FirstNameList(0) = "John"
FirstNameList(1) = "Jeffrey"
FirstNameList(2) = "Anna"
FirstNameList(3) = "Monica"
FirstNameList(4) = "Lisa"
FirstNameList(5) = "Steve"
AgeList(0) = "25"
AgeList(1) = "44"
AgeList(2) = "38"
AgeList(3) = "17"
AgeList(4) = "32"
AgeList(5) = "59"
NumLigne = DFLShowData3(6, "NAME", NameList, "FIRSTNAME", FirstNameList, "AGE", AgeList)
```

6.4.5.10 DFLShowData4

Displays a window with a list of propositions , which is filled from the *Propositions1* to *Propositions4* arguments.

The *Propositions1, Propositions2, Propositions3* and *Propositions4* parameter are Variant but must be declared as String tables, otherwise an error message appears when calling `DFLShowData4`.

Return Value: Long
The index of the selected line, or -1 if the user used the **ESC** key

Syntax

Visual Basic

`ShowData4(PropCountMax As String, ColTitle1 As String, Propositions1 As Variant, ColTitle2 As String, Propositions2 As Variant, ColTitle3 As String, Propositions3 As Variant, ColTitle4 As String, Propositions4 As Variant) As Long`

Details

Table 6-81: Parameters

Parameter	Description
PropCountMax As Integer	Maximum number of propositions displayed on the screen
ColTitle1 As String	Title of the first column of the propositions list
Propositions1(N) As String	List of values of the first column of the propositions list
ColTitle2 As String	Title of the second column of the propositions list
Propositions2(N) As String	List of values of the second column of the propositions list
ColTitle3 As String	Title of the third column of the propositions list
Propositions3(N) As String	List of values of the third column of the propositions list
ColTitle3 As String	Title of the fourth column of the propositions list
Propositions3(N) As String	List of values of the fourth column of the propositions list

Example:

```
Dim NameList(5) As String
Dim FirstNameList(5) As String
Dim AgeList(5) As String
Dim TownList(5) As String
NameList(0) = "McClain"
NameList(1) = "Kennedy"
NameList(2) = "Smith"
NameList(3) = "Nicols"
NameList(4) = "Jordan"
NameList(5) = "Bryant"
FirstNameList(0) = "John"
FirstNameList(1) = "Jeffrey"
FirstNameList(2) = "Anna"
FirstNameList(3) = "Monica"
FirstNameList(4) = "Lisa"
FirstNameList(5) = "Steve"
AgeList(0) = "25"
AgeList(1) = "44"
AgeList(2) = "38"
AgeList(3) = "17"
AgeList(3) = "17"
AgeList(4) = "32"
AgeList(5) = "59"
TownList(0) = "New-York"
TownList(1) = "San Diego"
TownList(2) = "Paris"
TownList(3) = "London"
TownList(4) = "Miami"
TownList(5) = "Memphis"
NumLigne = DFLShowData4(6, "NAME", NameList, "FISTNAME", FirstNameList, "AGE", AgeList,
"TOWN", TownList )
```

6.4.5.11 DFLShowDataN

Displays a window with a list of propositions , which is filled from the Propositions argument.

This function generalizes the option box with N columns. For the titles or the data, the columns must be separated by commas. In a case where a title or a column contains a space or a comma, you must put this datum inside double quotes " (Chr(34)).

The *Propositions* parameter is Variant but must be declared as a String table, otherwise, an error message appears when calling `DFLShowDataN`.

Return Value: Long

The index of the selected line, or -1 if the user used the **ESC** key

Syntax

Visual Basic

`DFLShowDataN(`*PropCountMax* `As String,` *ColsTitle* `As String,` *Propositions* `As Variant) As Long`

Details

Table 6-82: Parameters

Parameter	Description
PropCountMax As Integer	The maximum number of propositions displayed on the screen
ColsTitle As String	It contains the title of columns to be displayed, under the format 'Title1, Title2 , Title3'
Propositions(N) As String	It contains the propositions list, under the format '"Col1[RowN]", "Col2[RowN]", "Col3[RowN]"' where Col1[RowN], Col2[RowN], Col3[RowN] are values of the different columns for the row N

Example:

```
Dim Options(2) As String
Dim Opt01(2) As String
Dim Opt02(2) As String
Dim Opt03(2) As String
Opt01(0) = "file1"
Opt02(0) = "file1"
Opt03(0) = "file1"
Opt01(1) = "dir1"
Opt02(1) = "dir2"
Opt03(1) = "dir3"
Opt01(2) = "drive1"
Opt02(2) = "drive2"
Opt03(2) = "drive3"
For i=0 To 2
Options(i)=Chr(34) + Opt01(i) + Chr(34) + "," + Chr(34) + Opt02(i) + Chr(34) + "," +
Chr(34) + Opt03(i) + Chr(34)
Next i
RET= DFLShowDataN(3, "Title1,Title2,Title3", Options)
```

6.4.5.12 DFLShowMessage

Return Value: Long

Returned value is always one.

Displays the message passed in the argument Mess in a message box with an **OK** button. This window disappears when the user selects **OK**.

Syntax

Visual Basic

```
DFLShowMessage(Mess As String) As Long
```

Details

Table 6-83: Parameters

Parameter	Description
Mess As String	Message/Question

Example:

```
OK = DFLShowMessage ("This message is shown in a window")
```

6.4.6 DFL TIFF Image Functions

Use the equivalent *VBA* function instead of the Dispatcher library function whenever possible.

Table 6-84: *TIFF* Image Functions

Function	VBA equivalent
DFLTIFFDrawText	-
DFLTIFFEraseBox	-
DFLTIFFExtractBox	-
DFLTIFFMonoTIFF	-
DFLTIFFMultiTIFF	-
DFLTIFFRes	-
DFLTIFFResChange	-
DFLTIFFRotate180	-
DFLTIFFRotateLeft	-
DFLTIFFRotateRight	-
DFLTIFFSize	-

6.4.6.1 DFLTIFFDrawText

Places text specified in *IText* on top of an image file *SourceImage*, using the specified *IFont*, *IFontSize* and position *XPos*, *YPos*. The resulting image is saved in *DestImage*.

Return Value: Boolean

True if the operation was completed successfully, False otherwise

Syntax

Visual Basic

```
DFLTIFFDrawText(SourceImage As String, DestImage As String, IText As String, IFont As String, IFontSize As Long, XPos As Long, YPos As Long) As Boolean
```

Details

Table 6-85: Parameters

Parameter	Description
SourceImage As String	Image full file name to be processed
DestImage As String	Image full file name resulting from the processing
IText As String	Image text to be written
IFont As String	Image font to write the text with
IFontSize As Long	Image font size to write the text with
XPos As Long	Horizontal coordinate (in pixels) for the left part of the text zone
YPos As Long	Vertical coordinate (in pixels) for the top part of the text zone

Example:

```
OK = DFLTIFFDrawText("D:\temp\SCAN0008.tif","D:\temp\res0008.tif","Copyright © 2018 Open Text","Arial",24,80,20)
```

Note: The list of fonts available to this function is the same as for Microsoft Windows. The color of the text is black.

6.4.6.2 DFLTIFFEraseBox

Enables you to erase or whiten a rectangle on a source image file, using the specified coordinates, height, and width in pixels.

Return Value: Boolean

True if the operation was completed successfully, False otherwise

Syntax

Visual Basic

```
DFLTIFFEraseBox(SourceImage AsSourceImagePathAndFileName: sString, XPos As Long: integer, YPos As: Longinteger, IHeight As Long: integer, IWidth As LongW: integer) AS-> Boolean
```

Details

Table 6-86: Parameters

Parameter	Description
SourceImage As String	Image full file name to be processed
XPos As Long	Horizontal coordinate (in pixels) for the left part of the erased/white zone
YPos As Long	Vertical coordinate (in pixels) for the top part of the erased/white zone
IHeight As Long	Image height (in pixels) of the erased/white zone
IWidth As Long	Image width (in pixels) of the erased/white zone

Example: *OK = DFLTIFFEraseBox("D:\temp\res0008.tif",20,80,50,100)*

6.4.6.3 DFLTIFFExtractBox

Enables you to select part of a *TIFF* image as indicated by the argument *SourceImagePathAndFileName* and to save it as a separate *TIFF* file, as indicated by the argument *TargetImagePathAndFileName*.

The selection is indicated by the arguments X, Y, coordinates for an origin, from the top left corner of the source file, and then the height H and width W of the selection (with the origin point X,Y being at the top left of the selection frame).

X, Y, H, and W are all expressed in pixels.

Return Value: Boolean
True if the source file exists. False otherwise.

Syntax

Visual Basic

```
DFLTIFFExtractBox(SourceImage As String, DestImage As String, Xpos As Long, YPos As Long, IHeight As Long, IWidth As Long) As Boolean
```

Details

Table 6-87: Parameters

Parameter	Description
SourceImage As String	Image full file name to be processed

Parameter	Description
DestImage As String	Image full file name resulting from the processing
XPos As Long	Horizontal coordinate (in pixels) for the left part of the text zone
YPos As Long	Vertical coordinate (in pixels) for the top part of the text zone
IHeight As Long	Images height (in pixels) of the erased/white zone
IWidth As Long	Image width (in pixels) of the erased/white zone

Example:

```
OK = DFLTIFFExtractBox("c:\source.tif","c:\target.tif",0,0,100,100)
```

Builds a square image `c:\target.tif` of 100 pixels wide from the top left of `c:\source.tif`.

6.4.6.4 DFLTIFFMonoTIFF

Enables transformation of a multi-image *TIFF* file into several mono-image *TIFF* files.

The target files will be named 01.TIF, 02.TIF or PREFIX01.TIF, PREFIX02.TIF.

Return Value: Boolean
True if the operation was completed successfully, False otherwise

Syntax

Visual Basic

`DFLTIFFMonoTIFF(`*`MultiTIFFSourceFullFileName`* `As String,`
`DestMonoTIFFDirName` `As String) As Boolean`

Details

Table 6-88: Parameters

Parameter	Description
MultiTIFFSourceFullFileName As String	Full file name of the multi-image source *TIFF* file

Parameter	Description
MonoTiffFileTargetFolder: String	The target directory in which you put the result of the split of the multi-image source *TIFF* file. If *TIFF* files of the same names already exist, they will be overwritten by the result of the split. You must finish your directory variable by the back slash sign \ or the last part will be taken as a prefix

Example:

```
DFLTIFFMonoTIFF("D:\Dispatcher\Repacq\MultiImageTiffFile.tif","D:\Dispatcher\DestImage")
Creates DestImage01.TIF, DestImage01.TIF in the directory D:\Dispatcher
```

Note: The destination directory is not created if it does not exist. Result files are not created and no error message is displayed.

6.4.6.5 DFLTIFFMultiTIFF

Enables transformation of several mono-Image *TIFF* files into a multi-Image *TIFF* file.

Return Value: Boolean

True if the operation was completed successfully, False otherwise

Syntax

Visual Basic

`DFLTIFFMultiTIFF(`*`MonoTIFFTab`* `As Variant, `*`ImagesCount`* `As Long, `*`DestMultiTIFFFullFileName`* `As String) As Boolean`

Details

Table 6-89: Parameters

Parameter	Description
MonoTIFFTab(N) As String	The list of the path and file names of the mono-image *TIFF* files The *MonoTIFFTab* parameter is a Variant but must be declared as a String table, otherwise an error message appears when calling `DFLTIFFMultiTIFF`
ImagesCount As Long	The number of elements of the *MonoTIFFTab*argument
DestMultiTIFFFullFileName As String	Full file name of the multi-image *TIFF* file to be created, overwriting existing file with the same name

Example:

```
Dim sTab(2) As String
stTab(0)="C:\Stock\Pic\Image1.tif"
stTab(1)="C:\Stock\Pic\Image2.tif"
stTab(2)="C:\Stock\Pic\Image3.tif"
DFLTIFFMultiTIFF(stTab,3,"C:\Stock\Pic\MultiImageTiffFile.tif")
```

6.4.6.6 DFLTIFFRes

Return Value: Long

Returns the vertical resolution of a specified *TIFF* file.

Syntax

Visual Basic

`DFLTIFFRes(`*`SourceImage`* `As String) As Long`

Details

Table 6-90: Parameters

Parameter	Description
SourceImage As String	Full file name of the image file

Example:

```
Dim res As Long
Res = DFLTIFFRes("D:\temp\SCAN0008.tif")
```

6.4.6.7 DFLTIFFResChange

Enables modification of the resolution of a *TIFF* image by creating an image with a new resolution (the horizontal and vertical resolutions are identical).

Return Value: Boolean

True if the operation was completed successfully, False otherwise

Syntax

Visual Basic

`DFLTIFFResChange(`*`InputFullFileName`* `As String,` *`OutputFullFileName`* `As String,` *`NewResolution`* `As Long) As Boolean`

Details

Table 6-91: Parameters

Parameter	Description
InputFullFileName As String	Full file name of the input file
OutputFullFileName As String	Full file name of the output file
NewResolution As Long	Resolution for the output file (in dpi)

Example:

```
DFLTIFFResChange("D:\Temp\ImgIn.tif", "D:\T\ImgOut.tif",200)
// the new resolution is 200 x 200 dpi
```

6.4.6.8 DFLTIFFRotate180

Applies a 180° rotation. The result of the rotation of the image contained in the SourceImage file is saved in the DestImage file.

Return Value: Boolean
True if the operation was completed successfully, False otherwise

Syntax

Visual Basic

```
DFLTIFFRotate180(SourceImage As String, DestImage As String) As Boolean
```

Details

Table 6-92: Parameters

Parameter	Description
SourceImage As String	Image full file name to be processed
DestImage As String	Image full file name resulting from the processing

Example:

```
OK = DFLTIFFRotate180("c:\source.tif","c:\dest.tif")
```

6.4.6.9 DFLTIFFRotateLeft

Applies a counter-clockwise 90° rotation. The result of the rotation of the image contained in the SourceImage file is saved in the DestImage file.

Return Value: Boolean
True if the operation was completed successfully, False otherwise

Syntax

Visual Basic

```
DFLTIFFRotateLeft(SourceImage As String, DestImage As String) As Boolean
```

Details

Table 6-93: Parameters

Parameter	Description
SourceImage As String	Image full file name to be processed
DestImage As String	Image full file name resulting from the processing

Example:

```
OK = DFLTIFFRotateLeft("c:\source.tif","c:\target.tif")
```

6.4.6.10 DFLTIFFRotateRight

Applies a clockwise 90° rotation. The result of the rotation of the image contained in the SourceImage file is saved in the DestImage file.

Return Value: Boolean
True if the operation was completed successfully, False otherwise

Syntax

Visual Basic

```
DFLTIFFRotateRight(SourceImage As String, DestImage As String) As
Boolean
```

Details

Table 6-94: Parameters

Parameter	Description
SourceImage As String	Image full file name to be processed
DestImage As String	Image full file name resulting from the processing

Example:

```
OK = DFLTIFFRotateRight("c:\source.tif","c:\target.tif")
```

6.4.6.11 DFLTIFFSize

Return Value: Boolean

The width and the height of a given TIFF image. True if the source file exists, False otherwise.

Syntax

Visual Basic

```
DFLTIFFSize(SourceImage : string, IWidth as Long, IHeight As Long) As Boolean
```

Details

Table 6-95: Parameters

Parameter	Description
SourceImage As String	Image full file name to be processed
IWidth As Long	Image width (in pixels)
IHeight As Long	Images height (in pixels)

Example:

```
Dim W As Long
Dim H As Long
OK = DFLTIFFSize("c:\source.tif", W, H)
```

6.4.7 DFL System Functions

Use the equivalent *VBA* function instead of the Dispatcher library function whenever possible.

Table 6-96: System Functions and VBA Equivalents

Function	VBA equivalent
"DFLCallDLL1" on page 230	-
"DFLCallDLL2" on page 231	-
"DFLCallExe" on page 232	-
"DFLCallExeAndWait" on page 232	-
"DFLGetHostName" on page 233	-
"DFLGetUserName" on page 233	-
"DFLSleep" on page 234	-

6.4.7.1 DFLCallDLL1

Calls a *DLL* function and passes an input argument.

Return Value: Boolean
True if the operation was completed successfully. False otherwise.

Syntax

Visual Basic

```
DFLCallDLL1(DllName As String, FunctionName As String, Input As String)
As Boolean
```

Details

Table 6-97: Parameters

Parameter	Description
DllName As String	Full access to *DLL* file name (Unicode-compliant)
FunctionName As String	Name of the function (ASCII characters only)
Input As String	Function input (Unicode-compliant)

Example:

```
OK = DFLCallDLL1("C:\myDLL.dll", "myFunction", "InputArgument")
```

The function that is called must correspond to the following:

- In Pascal:

```
function myFunction(input: PChar):boolean; stdcall;
```

- In C:

```
__stdcall bool myFunction (char* input);
```

6.4.7.2 DFLCallDLL2

Calls a *DLL* function (with one input and one output) and passes an input argument. It places the results of the call into the argument Output.

Return Value: Boolean
True if the call was successful. False otherwise.

Syntax

Visual Basic

`DFLCallDLL2(`*`DllName`* `As String,` *`FunctionName`* `As String,` *`Input`* `As String,` *`Output`* `As String) As Boolean`

Details

Table 6-98: Parameters

Parameter	Description
DllName As String	Full access to *DLL* file name (Unicode-compliant)
FunctionName As String	Name of the function (ASCII characters only)
Input As String	Function input (Unicode-compliant)
Output As String	Function output (Unicode-compliant)

Example:

```
Dim stMyResult As String
OK = DFLCallDLL2("c:\myDLL.dll", "myFunction", "InputArgument", stMyResult)
```

The function that is called must correspond to the following:

- In Pascal:

```
function myFunction (input:pChar; output:PChar; pSize:pLongInt) :boolean; stdcall;
```

- In C:

```
__stdcall bool myFunction(char* input, char* output, int * pSize);
```

6.4.7.3 DFLCallExe

This function runs an EXE file with a list of parameters.

Return Value: Boolean
`DFLCallExe` returns True if the EXE file is run, regardless of whether the operation succeeds or not.

Syntax

Visual Basic

```
DFLCallExe(ExeName As String, Params As String)As Boolean
```

Details

Table 6-99: Parameters

Parameter	Description
ExeName As String	Path and file name of the program to run
Params As String	Parameters to transmit to the programs in a command line

Example:

```
OK = DFLCallExe("notepad.exe", "c:\temp\test.txt")// open the file "c:\temp\test.txt" in
Notepad.
```

Note: Include the full path of the EXE file, unless you have declared these programs or their folders as public in Windows.

6.4.7.4 DFLCallExeAndWait

Runs an EXE file with a list of parameters.

Unlike `CallExe`, `CallExeAndWait` will wait for the called program to exit prior to continuing the script.

Return Value: Boolean
True if the EXE file is run, regardless of whether the operation succeeds or not.

Syntax

Visual Basic

```
DFLCallExeAndWait(ExeName As String, Params As String)As Boolean
```

Details

Table 6-100: Parameters

Parameter	Description
ExeName As String	Path and file name of the program to run
Params As String	Parameters to transmit to the program, as in a command line

Example:

```
OK = DFLCallExeAndWait("notepad.exe", "c:\temp\test.txt") // open the file "c:\temp
\test.txt" in Notepad.
```

Use the full path of the EXE file, unless you have declared these programs or their folders as public in Windows.

6.4.7.5 DFLGetHostName

Return Value: String
The name of the current machine.

Syntax

Visual Basic

```
DFLGetHostName() As String
```

Details

PARAMETERS

None

Example:

```
DFLGetHostName() -> "MyComputer"
```

6.4.7.6 DFLGetUserName

Return Value: String
The current user name.

Syntax

Visual Basic

```
DFLGetUserName()As String
```

Details

PARAMETERS

None

Example:

```
DFLGetUserName() -> "Admin"
```

6.4.7.7 DFLSleep

The rest of the script is delayed by the number of specified milliseconds.

After this time is passed, the function returns true. However, it is not necessary to test this return value. A simple call such as Sleep(5000) is sufficient.

Return Value: Boolean

Not necessary to check. True if the operation was completed successfully, False otherwise.

Syntax

Visual Basic

```
DFLSleep(Ms As Long) As Boolean
```

Details

Table 6-101: Parameters

Parameter	Description
Ms As *Long*	Number of milliseconds to delay rest of script by.

Example:

```
DFLSleep(5000) //delays the rest of the script for 5 seconds
```

6.4.8 DFL File Management Functions

Use the equivalent *VBA* function instead of a Dispatcher library function whenever possible.

Table 6-102: File Management and *VBA* Equivalents

Function	VBA equivalent
DFLDirExists	-
DFLDirFileScan	-
DFLDiskFreeSpace	-
DFLDiskSize	-
???	FileCopy
DFLFileDate	-
DFLFileDelete	Kill(S)
DFLFileDir	-
DFLFileExists	-
DFLFileExt	-
DFLFileGetAttributes	-
DFLFileIsReadOnly	-
DFLFileLoad	-
DFLFileName	-
DFLFileRename	Name S1 as S2
DFLFileSave	-
DFLFileSetAttributes	-
DFLFileSize	FileLen(S)
DFLMakeDir	Mkdir(S)

6.4.8.1 DFLDirExists

Indicates if a directory specified by the argument DirName exists.

Return Value: Boolean
True if the directory exists, False otherwise

Syntax

Visual Basic

```
DFLDirExists(DirName As String) As Boolean
```

Details

Table 6-103: Parameters

Parameter	Description
DirName As String	Complete path of the directory to check the existence of

Example:

```
OK = DFLDirExists("C:\Temp")
```

6.4.8.2 DFLDirFileScan

Returns the number of files found in a given directory with a specified extension. It includes files found in subfolders if the boolean argument SubDir is set to true.

This function updates the list Answers with the path and file name of the found files, up to the argument AnswersCountMax .

Return Value: Long

The number of files with the extension ".txt" present in the directory *<[d: |Dispatcher]>* and its subdirectories.

Syntax

Visual Basic

```
DFLDirFileScan(DirName As String, Extension As String, SubDir As Boolean,
Answers As Variant, AnswersCountMax As Long) As Long
```

Details

Table 6-104: Parameters

Parameter	Description
DirName As String	directory name
Extension As String	File extension
SubStr As Boolean	Indicates if recursive search in subfolders
Answers(N) As String	Found files table *Answers* is Variant but must be declared as a table of String otherwise there is an error message when calling `DFLDirFileScan`.
AnswersCountMax As Long	Number max of found files

Example:

```
Dim stList(100) As String
Dim iNbFolder As Long
Dim bSubFolder As Boolean
bSubFolder=True
iNbFolder=DFLDirFileScan("d:\Dispatcher","*.txt",bSubFolder,stList,100)
```

6.4.8.3 DFLDiskFreeSpace

Returns the number of bytes of free space available on a specified drive.

Return Value: Long
Where 1 = A, 2 = B, 3=C, etc. Returns -1 if the disk number is incorrect.

Syntax

Visual Basic

`DFLDiskFreeSpace(`*`DiskNum`* `As Long) As Long`

Details

Table 6-105: Parameters

Parameter	Description
DiskNum As Long	Disk number

Example:

```
I = DFLDiskFreeSpace(3) // Gives the free space of "C:" drive
```

6.4.8.4 DFLDiskSize

Returns the size in bytes of a specified drive.

Return Value: Long
Where 1 = A, 2 = B, 3=C, etc. Returns -1 if the disk number is incorrect.

Syntax

Visual Basic

`DFLDiskSize(`*`DiskNum`* `As Long) As Long`

Details

Table 6-106: Parameters

Parameter	Description
DiskNum As Long	Disk number

Example:

```
I = DFLDiskSize(3) // Gives the size of "C:" drive
```

6.4.8.5 DFLFileCopy

Copies the content of any existing file in a new file. You can specify the name and path of the new file. If the new file already exists, the function overwrites the existing file.

Return Value: Boolean

True if the copy was completed successfully. False otherwise.

Syntax

Visual Basic

```
DFLFileCopy(OldFileName, NewFileName As String) As Boolean
```

Details

Table 6-107: Parameters

Parameter	Description
OldFileName As String	Initial full file name
NewFileName As String	New full file name

Example:

```
OK = DFLFileCopy("C:\InitialFile.txt", "C:\tmp\NewFileName.txt")
```

6.4.8.6 DFLFileDate

Return Value: String

The timestamp of the specified file. If the file does not exist, returns an empty string.

Syntax

Visual Basic

```
DFLFileDate(FullFileName As String) As String
```

Details

Table 6-108: Parameters

Parameter	Description
FullFileName As String	Full file name

Note: The date and time formats are controlled by the **Regional Settings** in the configuration panel of Windows. To open this element of the configuration panel, select **Start** > **Settings** > **Configuration Panel**, then double-click on the **Regional Settings** icon.

Example:

```
DFLFileDate("C:\File.txt") -> "03/02/2003 11:50:26"
```

6.4.8.7 DFLFileDelete

Deletes the file indicated in the argument *FullFileName* from the disk.

Return Value: Boolean
True if the operation was completed successfully; False otherwise

Syntax

Visual Basic

```
DFLFileDelete(FullFileName As String) As Boolean
```

Details

Table 6-109: Parameters

Parameter	Description
FullFileName As String	Full file name

Example:

```
OK = DFLFileDelete("C:\File.txt")
```

6.4.8.8 DFLFileDir

Return Value: String
The path of the file.

Extracts the path of the file from the string argument *FullFileName*.

Syntax

Visual Basic

```
DFLFileDir(FullFileName As String) As String
```

Details

Table 6-110: Parameters

Parameter	Description
FullFileName As String	Full file name

Example:

```
DFLFileDir("C:\directory\File.txt") -> "C:\directory"
```

6.4.8.9 DFLFileExists

Indicates if a file specified by the argument *FullFileName* exists.

Return Value: Boolean
True if the file exists; False otherwise

Syntax

Visual Basic

```
DFLFileExists(FullFileName As String) As Boolean
```

Details

Table 6-111: Parameters

Parameter	Description
FullFileName As String	Full file name

Example:

```
bFileExist = DFLFileExists("C:\Autoexec.bat")
```

6.4.8.10 DFLFileExt

Return Value: String
The extension part of the file name.

Syntax

Visual Basic

```
DFLFileExt(FullFileName As String) As String
```

Details

Table 6-112: Parameters

Parameter	Description
FullFileName As String	Full file name

Example:

```
DFLFileExt("C:\directory\File.txt") -> ".txt"
```

6.4.8.11 DFLFileGetAttributes

Returns -1 if an error occurred.

Return Value: Long
Returns the attributes of the specified file as an integer. This integer is unique and based on the addition of the values of each of the following attributes if the file combines several attributes:

Table 6-113: List of the possible attributes and their values

Attribute	Hexadecimal value	Decimal value	Description
ReadOnly	$00000001	1	The file is read-only
Hidden	$00000002	2	The file is hidden
SysFile	$00000004	4	The file is a system file
VolumeID	$00000008	8	Attribute deprecated
Directory	$00000010	16	The file is a directory
Archive	$00000020	32	The file is archived
AnyFile	$0000003F	71	All files

Syntax

Visual Basic

```
DFLFileGetAttributes(FullFileName As String) As Long
```

Details

Table 6-114: Parameters

Parameter	Description
FullFileName As String	Full file name

Example:

```
I = DFLFileGetAttributes("C:\File.txt")
```

6.4.8.12 DFLFileIsReadOnly

Return Value: Boolean
The read only status of the specified file.

Syntax

Visual Basic

```
DFLFileIsReadOnly(FullFileName As String) As Boolean
```

Details

Table 6-115: Parameters

Parameter	Description
FullFileName As String	Full file name

Example:

```
OK = DFLFileIsReadOnly("C:\file.txt")
```

6.4.8.13 DFLFileLoad

Return Value: Long
The number of elements that were successfully loaded in the list.

Loads the elements from a file specified in the argument *FileToLoad* (Path and File Name) into the argument *LoadingElt*.

Syntax

Visual Basic

`DFLFileLoad(`*FileToLoad* `As String, `*LoadingElt* `As Variant) As Long`

Details

Table 6-116: Parameters

Parameter	Description
FileToLoad As String	Full file name to load (Unicode-compliant)
LoadingElt(N) As Variant	FileToLoad content (ASCII characters only) The *LoadingElt* parameter is Variant but must be declared as a String table and dimensions of the table must be set before calling `DFLFileLoad` (otherwise an error message appears).

Example:

```
DFLFileLoad("c:\data\DataFile.txt", AnswerList)
// returns the number of elements loaded in the list AnswerList.
```

Example: Establish the Dimension of the Table and Then Redimension Before Calling the Function:

```
Dim LenghtArrayString As Long
Dim FileToLoad as String
Dim LoadingElt() as String
Dim Element As String
LenghtArrayString = 0
Open  FileToLoad  For Input As #1
While Not EOF(1)
     Line Input#1,Element
     LenghtArrayString = LenghtArrayString + 1
Wend
Close #1
ReDim LoadingElt(LenghtArrayString)
DFLFileLoad(FileToLoad,LoadingElt)
```

6.4.8.14 DFLFileName

Return Value: String
The name and extension of the file.

Extracts the name and extension of the file from the string argument *FullFileName*.

Syntax

Visual Basic

```
DFLFileName(FullFileName As String) As String
```

Details

Table 6-117: Parameters

Parameter	Description
FullFileName As String	Full file name

Example:

```
DFLFileName("C:\directory\File.txt") -> "File.txt"
```

6.4.8.15 DFLFileRename

Renames the file indicated by the argument *OldFileName* with the name indicated by the argument *NewFileName* .

Both arguments *OldFileName* and *NewFileName* should give the full path and file name.

Return Value: Boolean
True if the operation was completed successfully; False otherwise

Syntax

Visual Basic

```
DFLFileRename(OldFileName As String, NewFileName As String) As Boolean
```

Details

Table 6-118: Parameters

Parameter	Description
OldFileName As String	Old full file name
NewFileName As String	New full file name

Example:

```
OK = DFLFileRename("C:\FormerName.txt", "C:\NewName.txt")
```

6.4.8.16 DFLFileSave

Return Value: Long
The number of elements effectively saved in the file.

Creates a file specified in *FileToCreate* (path and file name) and writes to it the test data contained in the table *EltToSave*, for the first *FileLinesCount* lines.

Syntax

Visual Basic

DFLFileSave(*FileToCreate* As String, *FileLinesCount* As Long, *EltToSave* As Variant) As Long

Details

Table 6-119: Parameters

Parameter	Description
FileToCreate As String	Full file name to create (Unicode-compliant)
FileLinesCount As Long	Number of lines to save
EltToSave(N) As Variant	String elements to save in the *FileToCreate* (ASCII characters only) The *EltToSave* parameter is Variant but must be declared as a String table, otherwise, an error message appears when calling DFLFileSave.

Example:

```
Dim AnswerList(4) As String
AnswerList(0) = "Kennedy"
AnswerList(1) = "Jordan"
AnswerList(2) = "Smith"
AnswerList(3) = "Nicols"
DFLFileSave("c:\data\FichierData.txt", 3, AnswerList)
// returns 3, the number of elements effectively saved in the file.
```

6.4.8.17 DFLFileSetAttributes

Sets the attributes of the specified file.

Table 6-120: List of the possible attributes and their values

Attribute	Hexadecimal value	Decimal value	Description
ReadOnly	$00000001	1	The file is read-only
Hidden	$00000002	2	The file is hidden
SysFile	$00000004	4	The file is a system file
VolumeID	$00000008	8	Attribute deprecated
Directory	$00000010	16	The file is a directory
Archive	$00000020	32	The file is archived
AnyFile	$0000003F	71	All files

Return Value: Boolean

True if the operation was completed successfully; False otherwise

Syntax

Visual Basic

`DFLFileSetAttributes(`*`FullFileName`* `As String,` *`Attributes`* `As Long) As Boolean`

Details

Table 6-121: Parameters

Parameter	Description
FullFileName As String	Full file name
Attributes As Long	File attributes (addition of the values if the file combines several attributes)

Example:

```
OK = DFLFileSetAttributes("C:\File.txt",33)
// sets the properties to ReadOnly and Archive
```

6.4.8.18 DFLFileSize

Return Value: Long
The size of the specified file, in bytes.

Syntax

Visual Basic

```
DFLFileSize(FullFileName As String) As Long
```

Details

Table 6-122: Parameters

Parameter	Description
FullFileName As String	Full file name

Example:

```
FSize = DFLFileSize("D:\temp.txt")
```

6.4.8.19 DFLMakeDir

Creates a directory and, if necessary, its parent folders.

Return Value: Boolean
True if all the necessary folders have been created; False otherwise.

Syntax

Visual Basic

```
DFLMakeDir(DirName As String) As Boolean
```

Details

Table 6-123: Parameters

Parameter	Description
DirName As String	Complete path of the directory to be created

Example:

```
OK = DFLMakeDir("C:\tmp\Dir1")
```

6.4.9 DFL CheckBox Functions

Use the equivalent *VBA* function instead of a Dispatcher library function whenever possible.

Table 6-124: Checkbox Functions and VBA Equivalents

Function	VBA equivalent
"DFLCheckBox0" on page 248	-
"DFLCheckBox0Indexed" on page 249	-
"DFLCheckBox2Indexed" on page 250	-
"DFLCheckBox2Indexed" on page 250	-
"DFLCheckBoxIndexed" on page 252	-

6.4.9.1 DFLCheckBox0

A checkbox is considered selected if it is different from zero.

Return Value: String

Returns the number of the first checkbox that is selected (between 1 and *CheckBoxCount*).

If no checkbox is selected, the function returns -1.

Syntax

Visual Basic

```
DFLCheckBox0(CheckBoxValues As Variant, CheckBoxCount As Long) As String
```

Details

Table 6-125: Parameters

Parameter	Description
CheckBoxValues As String(N)	Check boxes values The *CheckBoxValues* parameter is a Variant but must be declared as a String table, otherwise, an error message appears when calling DFLCheckBox0.
CheckBoxCount As Long	Check boxes count

Example:

```
Dim CASES(3) As String
CASES(0) = "O"
```

```
CASES(1) = "0"
CASES(2) = "1"
CASES(3) = "0"
DFLCheckBox0(CASES, 4) -> returns "3"
```

6.4.9.2 DFLCheckBox0Indexed

A checkbox is considered selected if it is different from zero. The argument *Idx* contains the values associated with each checkbox.

Return Value: String

Returns the character within the argument Index corresponding to the first checkbox that is selected (between 1 and *CheckBoxCount*).

If no checkbox is selected, the function returns a space "" or the default value read in the argument *Idx*, corresponding to *CheckBoxCount* + 1.

Syntax

Visual Basic

`DFLCheckBox0Indexed(`*CheckBoxValues* `As Variant,` *CheckBoxCount* `As Long,` *Idx* `As String) As String`

Details

Table 6-126: Parameters

Parameter	Description
CheckBoxValues As String(N)	Check boxes values The *CheckBoxValues* parameter is a Variant but must be declared as a String table, otherwise, an error message appears when calling `DFLCheckBox0Indexed`.
CheckBoxCount As Long	Check boxes count
Idx As String	Indexes associated with each checkbox

Example:

```
Indexes associated with each checkbox
Dim CASES(3) As String
CASES(0) = "0"
CASES(1) = "0"
CASES(2) = "1"
CASES(3) = "0"
DFLCheckBox0Indexed(CASES, 4, "ABCD") returns "C"
CASES(0) = "0"
CASES(1) = "0"
CASES(2) = "0"
CASES(3) = "0"
DFLCheckBox0Indexed(CASES, 4, "ABCD") returns " "
CASES(0) = "0"
CASES(1) = "0"
```

```
CASES(2) = "O"
CASES(3) = "O"
DFLCheckBoxOIndexed(CASES, 4, "ABCDE") returns "E"
```

6.4.9.3 DFLCheckBox2Indexed

A checkbox is considered selected if it is different from zero. The argument Index contains the values associated with each checkbox.

Return Value: String

- If more than two checkboxes are selected, the function returns *OtherAnswer*.
- If two checkboxes are selected, the function returns the values associated in each case with *AnswerIndex* .
- If only one checkbox is selected, the function returns the value associated with the checkbox followed by the default value.
- If no checkboxes are selected, the function returns twice the default value.

The default value is calculated as follows: if the number of characters in AnswerIndex is superior to the number of checkboxes, the default value is the character at the position (*CheckBoxCount* + 1) in AnswerIndex , otherwise the function returns the space character “”.

Syntax

Visual Basic

`DFLCheckBox2Indexed(`*CheckBoxValues* `As Variant,` *CheckBoxCount* `As Long,` *AnswerIndex* `As String,` *OtherAnswer* `As String) As String`

Details

Table 6-127: Parameters

Parameter	Description
CheckBoxValues As String(N)	Check boxes values
CheckBoxCount As Long	Check boxes count
AnswerIndex As String	Indexes associated with each checkbox The argument *AnswerIndex* contains the values associated with each checkbox, eventually followed by a value for the default.

Parameter	Description
OtherAnswer As String	Returned value if more than 2 Checkboxes are checked The argument *OtherAnswer* contains the string to be returned if strictly more than two checkboxes are checked.

Example:

```
Dim CASES(3) As String
CASES(0) = "1"
CASES(1) = "1"
CASES(2) = "1"
CASES(3) = "0"
DFLCheckBox2Indexed(CASES, 4, "ABCDZ", "0") returns "0"
CASES(0) = "1"
CASES(1) = "0"
CASES(2) = "1"
CASES(3) = "0"
DFLCheckBox2Indexed(CASES, 4, "ABCDZ", "0") returns "AC"
CASES(0) = "0"
CASES(1) = "0"
CASES(2) = "1"
CASES(3) = "0"
DFLCheckBox2Indexed(CASES, 4, "ABCDZ", "0") returns "CZ"
CASES(0) = "0"
CASES(1) = "0"
CASES(2) = "1"
CASES(3) = "0"
DFLCheckBox2Indexed(CASES, 4, "ABCD", "0") returns "C " i.e. "C" followed by a space
CASES(0) = "0"
CASES(1) = "0"
CASES(2) = "0"
CASES(3) = "0"
DFLCheckBox2Indexed(CASES, 4, "ABCDZ", "0") returns "ZZ"
CASES(0) = "0"
CASES(1) = "0"
CASES(2) = "0"
CASES(3) = "0"
DFLCheckBox2Indexed(CASES, 4, "ABCD", "0") returns " " i.e. 2 spaces
```

Note: The *CheckBoxValues* parameter is a Variant but must be declared as a String table, otherwise, an error message appears when calling `DFLCheckBox2Indexed`.

6.4.9.4 DFLCheckBoxEmpty

A checkbox is considered selected if it is different from an empty string.

Return Value: String

It returns the number of the first checkbox that is selected (between 1 and *CheckBoxCount*).

If no checkbox is checked, the function returns -1.

Syntax

Visual Basic

```
DFLCheckBoxEmpty(CheckBoxValues As Variant, CheckBoxCount As Long) As String
```

Details

Table 6-128: Parameters

Parameter	Description
CheckBoxValues As String(N)	Check boxes values The *CheckBoxValues* parameter is a Variant but must be declared as a String table, otherwise, an error message appears when calling DFLCheckBoxEmpty.
CheckBoxCount As Long	Check boxes count

Example:

```
Dim CASES(4) As String
CASES(0) = ""
CASES(1) = ""
CASES(2) = "A"
CASES(3) = ""
DFLCheckBoxEmpty(CASES, 4) returns "3"
```

6.4.9.5 DFLCheckBoxIndexed

A checkbox is considered selected if it is different from the empty string. The argument Index contains the values associated with each checkbox.

Return Value: String

It returns the character within the argument Index corresponding to the first selected checkbox (between 1 and *CheckBoxCount*).

If no checkbox is selected, the function returns a space "" or the default value read in the argument Index, corresponding to *CheckBoxCount* + 1.

Syntax

Visual Basic

```
DFLCheckBoxIndexed(CheckBoxValues As Variant, CheckBoxCount As Long, Idx As String) As String
```

Details

Table 6-129: Parameters

Parameter	Description
CheckBoxValues As String(N)	Check boxes values The parameter *CheckBoxValues* is a Variant but must be declared as a String table, otherwise, an error message appears when calling `DFLCheckBoxIndexed`
CheckBoxCount As Long	Check boxes count
Idx As String	Indexes associated with each checkbox

Example:

```
Dim CASES(3) As String
CASES(0) = ""
CASES(1) = ""
CASES(2) = "A"
CASES(3) = ""
DFLCheckBoxIndexed(CASES, 4, "ABCD") returns "C"
CASES(0) = ""
CASES(1) = ""
CASES(2) = ""
CASES(3) = ""
DFLCheckBoxIndexed(CASES, 4, "ABCD") returns " "
CASES(0) = ""
CASES(1) = ""
CASES(2) = ""
CASES(3) = ""
DFLCheckBoxIndexed(CASES, 4, "ABCDE") returns "E"
```

6.5 Dispatcher Event Model

Since version 4.6, users can use the features of the Event Model to write *VBA* scripts to control Dispatcher data and objects and customize the classification and recognition steps.

An event is an action recognized by an object and to which the object can programmatically respond. Events are triggered at the document level every time a document is processed. Each event fires the methods or properties defined within that event at a particular stage within the application.

The script developer can implement field controls by determining:

- Which fields need to be controlled
- The sequence order of the different field controls
- How to implement intra field and inter field controls

Event model scripts are available in the following applications:

- Recognition Designer. Events are triggered in the test windows of **Template Test** and **Field Unit Test** (index and table field tests, not anchor tests) modules of this application.
- Other modules such as **Classification Test**, **HPA Test**, **Full text Rules**, **Anchor Unit Test** are not concerned by the Event Model.

These scripts are not available in the Free Form Designer module.

All production modules carry out the same processing in each product apart from certain minor differences in the life cycle of the Dispatcher project, which is further detailed in the different event definitions and configurations within this section.

6.5.1 Writing Scripts for the Event Model

The VBA language is an event driven programming language, and the first step in developing an event-driven program is to write a series of subroutines, or methods, called event-handler routines. These routines handle the events that the main program generates.

In each Recognition project, scripts are used according to the type of event they apply to:

- Project events (one script per project)
- Index family events (one script per family)

These scripts can then use external modules. Each script has the BAS file extension.

6.5.1.1 Using Scripts

In the Dispatcher Event Model, there is:

- One script per project: `Project.bas` saved to the `Resources` directory. This script contains all the handler subroutines for the project events.
- One script per index family: `Indexing_Family_Name.bas` saved to the `IdxClasses` directory. This script contains all the handler subroutines for the index family events.

6.5.1.1.1 Content

Like every VBA script, the project and index family scripts are composed of:

- A header containing the declaration of variables
- A body containing:
  - Predefined subroutines, each of one handling a specific event
  - Custom subroutines

6.5.1.1.2 Project Script File

The project script file contains all the handler subroutines for the project events. This file is named `Project.bas` and is located in the `\Resources\Scripts` directory of the project.

By default, the project script is available from each index family script, hence project script public variables or subroutines can be used or called in all index family scripts.

6.5.1.1.3 Index Family Script File

Each index family script file contains all handler subroutines for events, for a specific index family.

The name of the script file corresponds to the index family (plus the BAS extension), and is stored in the `IdxClasses` directory, like the index family file.

6.5.1.1.4 User-Defined Script Files

User-defined script files are used by project and/or index family scripts.

The script developer can use as many external scripts as required and can give the scripts any name.

Scripts may be stored either in the standard script directory (`Resources\Scripts` from *`<[Project]>`* directory) or in another custom directory.

The VBA '#uses comment indicates that a script uses symbols declared in another script module:

- `*#uses "<script file name>"` to use a script in the standard script directory
- `#uses "<script full path>"`or "`<script relative path>`") to use a script stored in a custom directory

Related Topics

"Using the VBA Script Editor" on page 159

"Editing a Project Script File" on page 161

"Using the VBA Script Editor" on page 159

"Using Module Scripts" on page 162

"Error Handling" on page 256

"Debugging" on page 256

6.5.1.2 Declarations

Dispatcher Event Model scripts are code modules, in other words, they implement a code library. A module may access other modules with its own #uses comments.

Note: VBA also includes class modules and object modules that implement an Active X Automation object. Advanced Recogntion uses code modules only.

Related Topics

"Using Module Scripts" on page 162

"Error Handling" on page 256

6.5.1.3 Error Handling

Errors can be application or script errors. Application errors cannot be identified or handled in scripts.

Script errors can be handled through the `On Error` instruction. Failing to implement the error handling system can cause the script to stop when an error occurs.

There are three syntax forms:

- `On Error GoTo 0`: Disable the error handler (default)
- `On Error GoTo label`: Send error conditions to an error handler
- `On Error Resume Next`: Error conditions continue execution at the next statement.

Note: An error handler is a subprogram that is called in response to an error event.

6.5.1.4 Debugging

The VBA Debug object is not supported by Advanced Recognition in production and has no effect. However, it can be used to help the developer while debugging in Recognition Designer.

Related Topics

"Using the VBA Script Editor" on page 159

"Using Module Scripts" on page 162

"Error Handling" on page 256

6.5.1.5 Life Cycle

Each script can be in one of the following six states:

- *Loading*: Script loaded from file into memory
- *Compiling*: Script syntax checked and translated
- *Initializing*: The script initialization header is run; global variables instantiated
- *Running*: Event code run
- *Pending*: Script is pending an event
- *Unloading*: Variables and script codes unloaded from memory

6.5.1.5.1 Project Script Life Cycle

The moment when the Project script is loaded depends on the context:

- In the Advanced Recognition production modules, the project script is loaded, compiled and initialized at project loading. The project script is unloaded at project unloading.
- In Recognition Designer, the project script is loaded, compiled and initialized as required by a test module (but not at project opening).

The project script is unloaded as the same time as the project (see Event sequences).

Note: The project script is kept in memory until it is unloaded, that is declared global variables are “persistent”throughout the life span of the project script.

6.5.1.5.2 Index Family Script Life Cycle

In the Advanced Recognition production modules and Recognition Designer, the index family script is loaded, compiled and initialized when the index family is loaded. In the same way, it is unloaded when the index family is unloaded.

As with the project script, the moment the index family script is loaded depends on the context:

- In production, each index family script is loaded (or unloaded) as soon as the action is performed on its associated family. Refer to loading indexing families and unloading indexing families) for more information on loading and unloading Index Families.

  Note: Each index family script is kept in memory until it is unloaded, that is declared global variables are “persistent” throughout the life span of the index family script. This means that if different indexing families are used during a process, all the index family scripts still run in parallel.
- In Recognition Designer, the index family script is loaded when needed (at the opening of the *GUI* script).

6.5.1.5.3 External Scripts

Every external script is loaded and initialized when it is called for the first time.

An external script is not reloaded if a second script uses it. The external script is persistent as long as an instance of a script using it is loaded. External scripts are unloaded as soon as they are no longer used.

6.5.1.6 Execution Errors

Two kinds of execution errors can occur:

6.5.1.6.1 Application Errors

These are errors generated by Advanced Recognition, such as an image file not found during processing, an incomplete project configuration, etc.

These kind of errors cannot be handled by scripts.

6.5.1.6.2 Script Errors

Script errors happen when any piece of script code:

- Accesses an object in a bad context
- Uses any VBA function which generates an error

In the case where such an error happens, the script is interrupted and the error is logged as all other Advanced Recognition errors. The process may decide to stop or not, depending on the context.

To avoid such situations, script errors can be handled anywhere they can happen in the script, by using the VBA error handling system:

Example:

```
On Error Goto <ErrorLabel>
<Piece of code>
        <ErrorLabel>:
<Error handling code>
Goto <ResumeLabel>
```

6.5.2 Using Events

Dispatcher events are composed of project events and index family events. For each event, information includes:

- When the event is triggered
- Which parameters (if any) are sent to or returned from the script
- The sequence(s) in which the event is triggered
- The module(s) in which the event is triggered

Project events

The following project events can be used in scripts for classification:

- "Project_Initialize" on page 260
- "Project_Finalize" on page 263
- "Project_BeginTask" on page 260
- "Project_EndTask" on page 262
- "Project_BeforeClassification" on page 261
- "Project_AfterClassification" on page 262
- "Project_BeforeDocumentClassification" on page 261
- "Project_AfterDocumentClassification" on page 261

Index family events

The following index family events can be used in scripts for recognition:

- "IndexingFamily_Initialize" on page 263
- "IndexingFamily_Finalize" on page 271
- "IndexingFamily_BeforeDocumentRecognition" on page 264
- "IndexingFamily_AfterDocumentRecognition" on page 265
- "<Field>_BeforeRecognition" on page 264
- "<Field>_AfterRecognition" on page 265
- "IndexingFamily_ControlDocument" on page 269
- "IndexingFamily_EnterDocument" on page 266
- "IndexingFamily_DocumentValidated" on page 269
- "IndexingFamily_EnterDocument" on page 266
- "IndexingFamily_EnterDocument" on page 266
- "<Table>_AfterDeleteRow" on page 269
- "<Table>_BeforeDeleteRow" on page 268
- "<Table>_AfterDeleteRow" on page 269
- "IndexingFamily_PressCustomHotKey" on page 270

The matrix of events lists the events together with their associated objects and the parameters (if any) sent to or returned from the script.

A module may be a complete application like Dispatcher Classification. Modules used by events include:

- The Classification module is used in production in Dispatcher

- The test modules used in Recognition Designer: Template Test and Unit Test

6.5.2.1 Project_Initialize

This event is triggered every time after a project has been successfully loaded. It is the first event to be triggered.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test)
- Classification

This event can be visualized in the following sequences:

- "Project Loading" on page 277
- "Template Test: Single Image Test" on page 282

6.5.2.2 Project_BeginTask

This event is triggered just before the beginning of the processing of a batch in the sequence.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test)
- Classification

This event can be visualized in the following sequences:

- "Classification" on page 273
- "Project Unloading" on page 278
- "Template Test: Single Image Test" on page 282
- "Unit Test: Single Image Test" on page 284

6.5.2.3 Project_BeforeClassification

This event is triggered when batch classification begins.

Parameters

None

Usage

This event is triggered in the following modules:

- Classification

This event can be visualized in the following sequences:

- "Classification" on page 273

6.5.2.4 Project_BeforeDocumentClassification

This event is triggered just before document classification starts.

Parameters

None

Usage

This event is triggered in the following modules:

- Classification

This event can be visualized in the following sequence:

- "Document Classification" on page 275

6.5.2.5 Project_AfterDocumentClassification

This event is triggered just after document classification has finalized.

Parameters

None

Usage

This event is triggered in the following modules:

- Classification

This event can be visualized in the following sequence:

- "Document Classification" on page 275

6.5.2.6 Project_AfterClassification

This event is triggered when batch classification finalizes.

Parameters

None

Usage

This event is triggered in the following modules:

- Classification

This event can be visualized in the following sequences:

- "Classification" on page 273

6.5.2.7 Project_EndTask

This event is triggered just after the end of the processing of a batch.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test)
- Classification

This event can be visualized in the following sequences:

- "Classification" on page 273
- "Project Unloading" on page 278
- "Template Test: Single Image Test" on page 282
- "Unit Test: Single Image Test" on page 284

6.5.2.8 Project_Finalize

This event is triggered every time a Recogntiion project is released in the Project unloading sequence.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test)
- Classification

This event can be visualized in the following sequences:

- "Project Unloading" on page 278
- "Template Test: Single Image Test" on page 282

6.5.2.9 IndexingFamily_Initialize

This event is triggered after the index family has been loaded.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test)
- Classification

Note: Pre-indexing enables running recognition during the Classification step.

This event can be visualized in the following sequences:

- "IndexingFamily_BeforeDocumentRecognition" on page 264
- "Template Test: Single Image Test" on page 282

6.5.2.10 IndexingFamily_BeforeDocumentRecognition

This event is triggered just before document recognition starts if recognition is performed on all fields (or on some fields if pre-indexing is implemented) but not when recognition is performed on one specific field.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test)
- Classification

This event can be visualized in the following sequences:

- "Document Pre-Indexing" on page 275
- "Unit Test: Single Image Test" on page 284

6.5.2.11 <Field>_BeforeRecognition

This event is triggered before field recognition starts.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test functionalities)
- Classification (if field is a pre-index field)

This event can be visualized in the following sequences:

- "Unit Test: Single Image Test" on page 284

6.5.2.12 <Field>_AfterRecognition

This event is triggered when field recognition finalizes but before field controls start. Notice that for a table field, this event is triggered only once after the recognition has been done on all line items of the table field, before the row construction.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test functionalities)
- Classification (if field is a pre-index field)

This event can be visualized in the following sequences:

- "Unit Test: Single Image Test" on page 284

6.5.2.13 IndexingFamily_AfterDocumentRecognition

This event is triggered when document recognition finalizes if recognition is performed on all fields (or on some fields if pre-indexing is implemented) but not when recognition is performed on one specific field.

This event is triggered before the control of the document.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test functionality)
- Classification (if pre-indexing is used)

This event can be visualized in the following sequences:

- "Document Pre-Indexing" on page 275
- "Unit Test: Single Image Test" on page 284

6.5.2.14 IndexingFamily_EnterDocument

This event is triggered each time a new document is selected during attended modules, either when the user has selected it or when selected automatically.

This event is always followed by the IndexingFamily_ExitDocument event.

Parameters

None

Usage

This event is triggered in the following module:

- Recognition Designer (Template Test functionality)

6.5.2.15 <Table>_BeforeAddRow

This event is triggered each time before a row is manually added/inserted in a table field.

Syntax

Visual Basic

```
<Table>_BeforeAddRow(CurrentRow, CanAdd)
```

Details

Table 6-130: Parameters

Name	Type	In/Out	Description	Default
CurrentRow	Integer	In	Number of rows to be inserted	
CanAdd	EOperationControl	Out	[ocAccept, ocAsk, ocRefuse] ocAccept/ ocRefuse: the row is inserted/ not inserted without confirmation ocAsk: a message window asks the user to confirm the insertion	ocAsk if it is a paragraph context, ocAccept otherwise

The *CanAdd* parameter gives the following insert behavior:

- Insert a row in a paragraph:
  - *CanAdd* = *ocAccept*: inserts a row in the current paragraph
  - *CanAdd* = *ocRefuse*: does not insert the row in a paragraph
  - *CanAdd* = *ocAsk*: displays the message "Insert a row in the current paragraph?". The user can choose between three responses:
    - [**Yes**]: inserts the row in the current paragraph
    - [**No**]: inserts the row in a new paragraph
    - [**Cancel**]: does not insert the row in a paragraph
- Other case:
  - *CanAdd* = *ocAccept*: inserts a row
  - *CanAdd* = *ocAsk*: displays the message "Insert a new row?". The user can choose between 2 responses: [**Yes**] or [**No**]
  - *CanAdd* = *ocRefuse*: does not insert the row

6.5.2.16 <Table>_AfterAddRow

This event is triggered each time after a row is manually added/inserted in a Table Field.

Syntax

Visual Basic

```
<Table>_AfterAddRow(CurrentRow)
```

Details

Table 6-131: Parameters

Name	Type	In/Out	Description
CurrentRow	Integer	In	Number of rows inserted

6.5.2.17 <Table>_BeforeDeleteRow

This event is triggered each time before a Table Field row is manually deleted. A script control can accept or not accept the deletion.

If the user deletes all the rows of a paragraph, this event is triggered only once and the *CurrentRow* parameter is the current row of the paragraph. In this case if:

- *CanDelete* = *ocAccept*: deletes all rows of current paragraph
- *CanDelete* = *ocAsk*: displays a window with the message "Delete all rows of current paragraph?". User have to choose between 2 responses: [**Yes**] [**No**]
- *CanDelete* = *ocRefuse*: does not delete rows of the current paragraph

Syntax

Visual Basic

```
<Table>_BeforeDeleteRow(CurrentRow, CanDelete)
```

Details

Table 6-132: Parameters

Name	Type	In/Out	Description	Default
CurrentRow	Integer	In	Number of rows to be deleted	
CanDelete	EOperationControl	Out	[ocAccept, ocAsk, ocRefuse] ocAccept/ ocRefuse: the row is deleted/ not deleted without confirmation ocAsk: a message window asks the user to confirm the deletion	ocAccept if all fields of currentRow are empty, ocAsk otherwise

6.5.2.18 <Table>_AfterDeleteRow

This event is triggered each time after the row has been deleted in a Table.

Syntax

Visual Basic

```
<Table>_AfterDeleteRow(CurrentRow)
```

Details

Table 6-133: Parameters

Name	Type	In/Out	Description
CurrentRow	Integer	In	Number of rows deleted

6.5.2.19 IndexingFamily_DocumentValidated

Before the control of a document, the document OK status is Unknown. After the control, the OK status is True if all the document fields are OK, False otherwise. This event is triggered each time a document OK switches from Unknown or False to True.

Parameters

None

Usage

This event is triggered in the following module:

- Recognition Designer (Template Test and Field Unit Test functionality)

6.5.2.20 IndexingFamily_ControlDocument

This is the main event for a document; it is triggered each time a control should be done on the document.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test functionalities)

- Classification (if pre-indexing is used)

This event can be visualized in the following sequences:

- "Table Wizard" on page 272
- "Document Pre-Indexing" on page 275
- "Recognition" on page 278
- "Row Addition" on page 280
- "Template Test: Single Image Test" on page 282
- "Unit Test: Single Image Test" on page 284
- "Row Deletion" on page 280

6.5.2.21 IndexingFamily_PressCustomHotKey

This event is triggered each time a custom hot keys sequence is done in an attended module, whether a specific field is focused or not.

Authorized custom hot keys are:

- Any function keys either separately or in combination with the **SHIFT** key or **CTRL** key.
- **CTRL** or **ALT** key in combination with any character on the keyboard.

Syntax

Visual Basic

```
IndexingFamily_PressCustomHotKey(KeyCode, Shift, Alt, Ctrl, Handled,
CurrentRow)
```

Details

Table 6-134: Parameters

Name	Type	In/Out	Description	Default
KeyCode	Long	In	Indicates the virtual key code pressed	
Shift	Boolean	In	Indicates that **SHIFT** key modifier is pressed	
Alt	Boolean	In	Indicates that **ALT** key modifier is pressed	

Name	Type	In/Out	Description	Default
Ctrl	Boolean	In	Indicates that **CTRL** key modifier is pressed	
Handled	Boolean	Out	Indicates that a Custom Hot Key is handled by the script code only.	False
CurrentRow	Integer	In	Index of the current row (-1 for an Index Field)	

This event can be visualized in the following sequence:

- Recognition Designer (Template Test functionality)

6.5.2.22 IndexingFamily_ExitDocument

This event is triggered each time a document is exited in an attended module, whether the last document was validated or not.

Parameters

None

Usage

This event is triggered in the following module:

- Recognition Designer (Template Test functionality)

6.5.2.23 IndexingFamily_Finalize

This event is triggered before the index family is unloaded from the project.

Parameters

None

Usage

This event is triggered in the following modules:

- Recognition Designer (Template Test and Field Unit Test)
- Classification (if pre-indexing is used)

This event can be visualized in the following sequences:

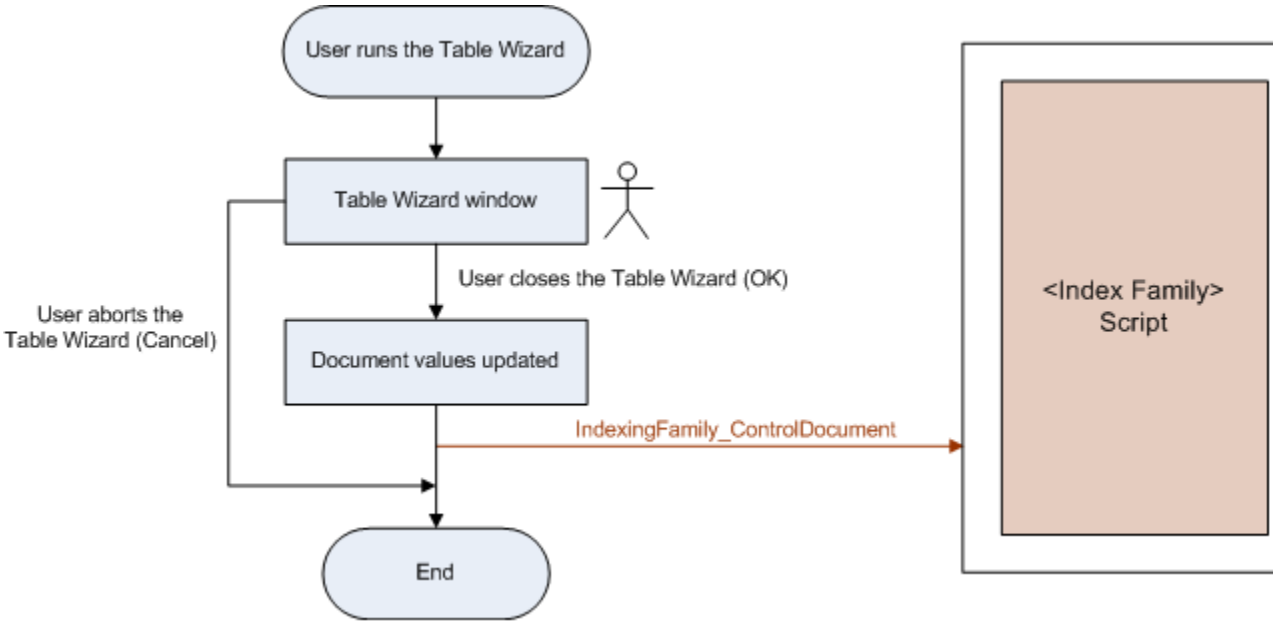
- "Unloading Indexing Families" on page 277
- "Template Test: Single Image Test" on page 282

6.5.3 Event Sequences

This section presents the different stages of processing and shows when different events are triggered within each of the following sequences:

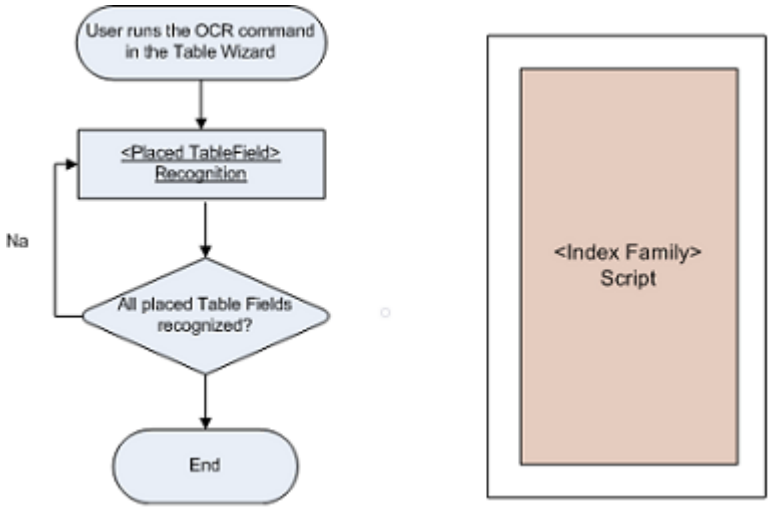
6.5.3.1 Table Wizard

The following diagram shows the event sequence triggered when the Table Wizard is used: IndexingFamily_ControlDocument.



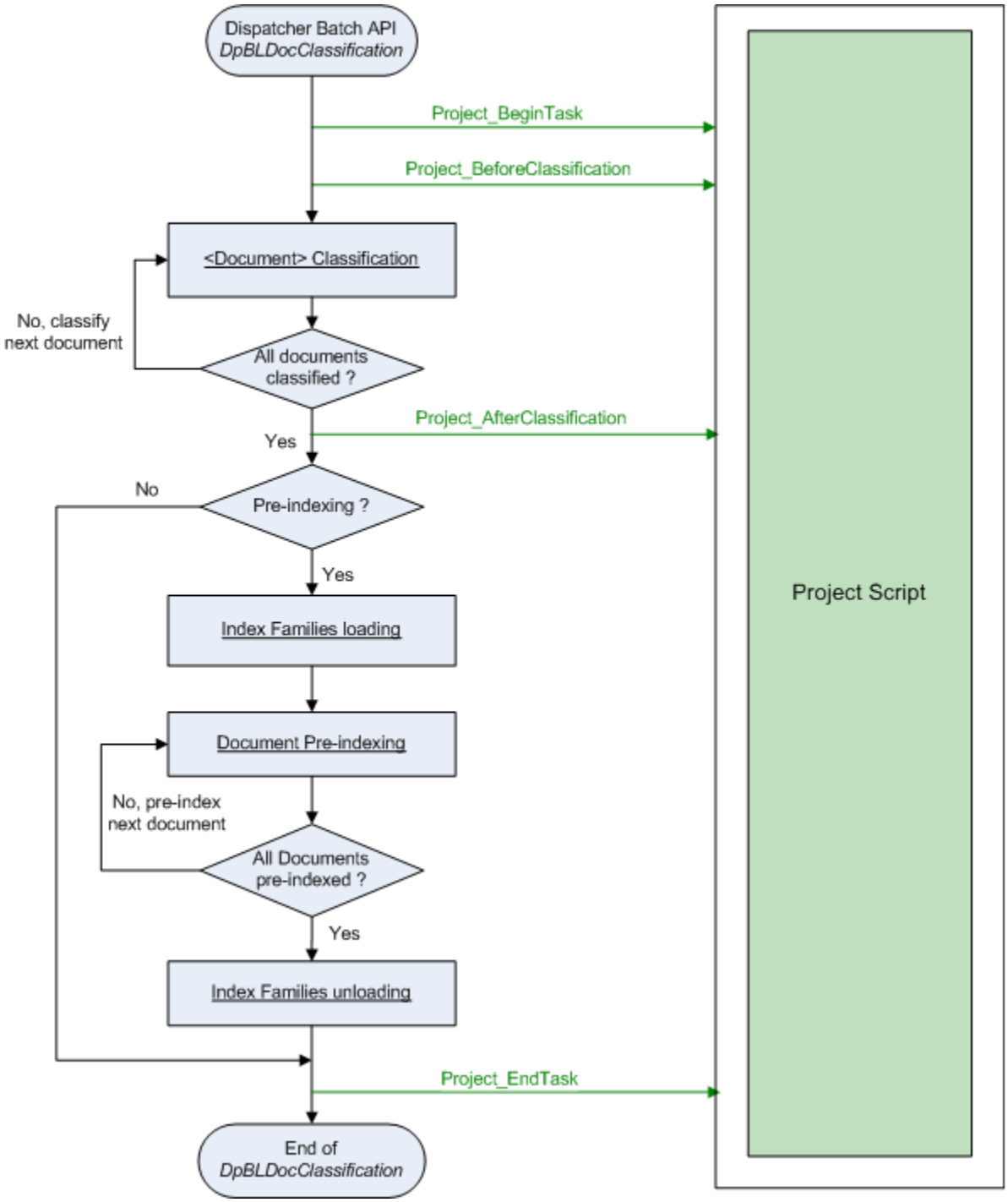
6.5.3.2 Table Wizard Recognition

This diagram shows the steps that are run when the user uses the *OCR* command in the Table Wizard:



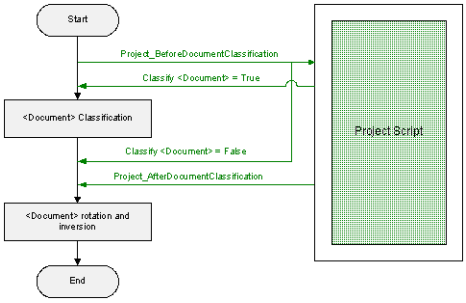
6.5.3.3 Classification

The following diagram shows the event sequence of the Classification production module.


Dispatcher Batch API
DpBLDocClassification
Project_BeginTask
Project_BeforeClassification
<Document> Classification
No, classify next document
All documents classified ?
Project_AfterClassification
Yes
No
Pre-indexing ?
Yes
Index Families loading
Document Pre-indexing
No, pre-index next document
All Documents pre-indexed ?
Yes
Index Families unloading
Project_EndTask
End of
DpBLDocClassification
Project Script

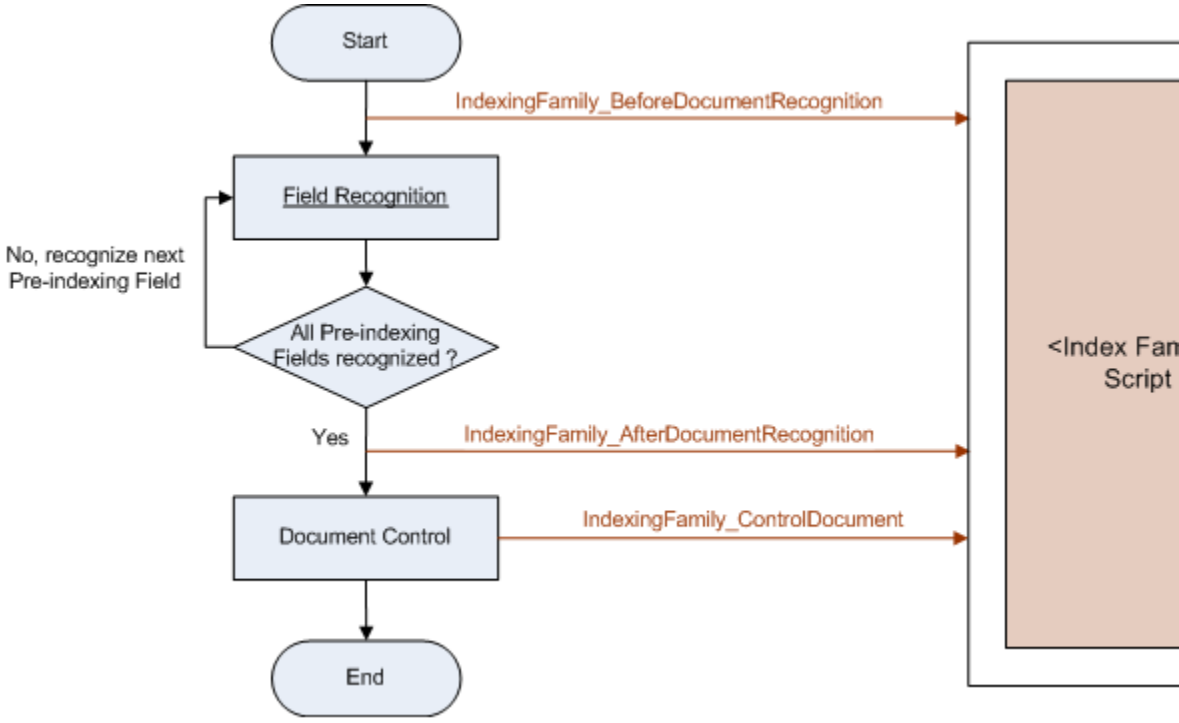
6.5.3.4 Document Classification

The following diagram shows the events triggered when a document (in single or duplex mode) is classified:



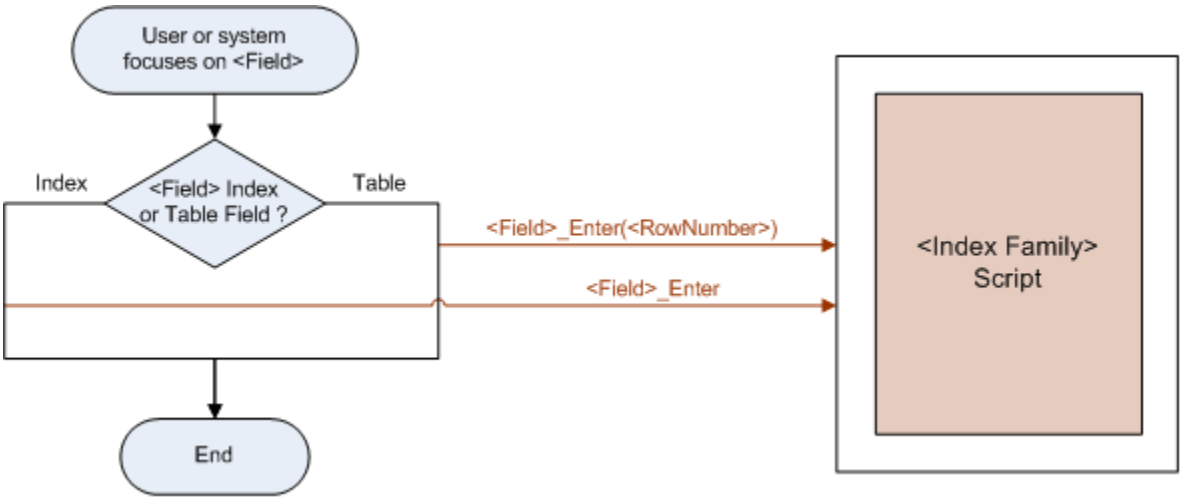
6.5.3.5 Document Pre-Indexing

The following diagram shows the event sequence for the pre-indexing (recognition + control of pre-index fields) of a single document:



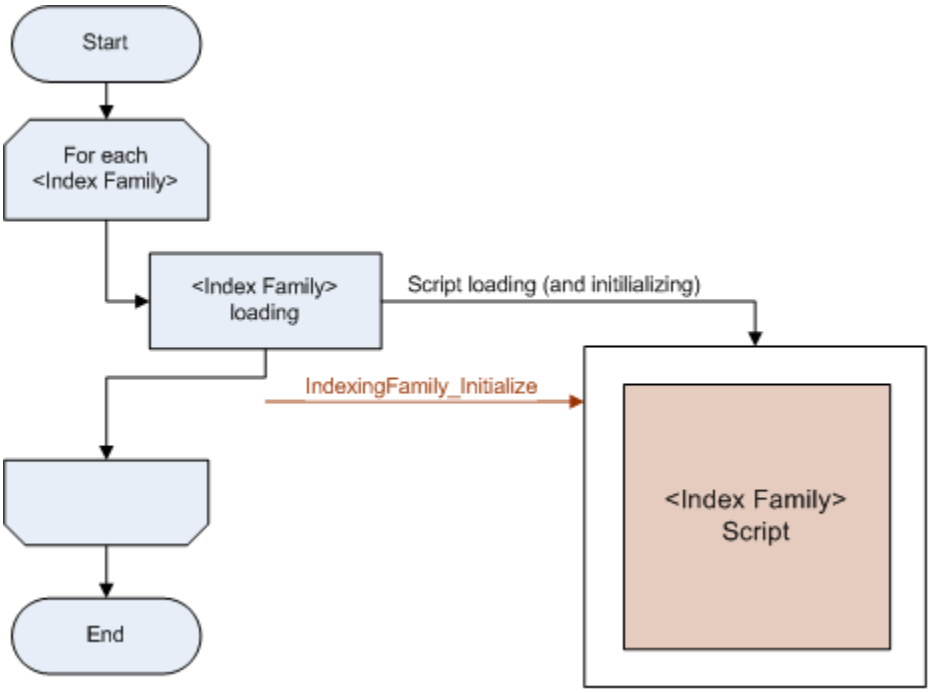
6.5.3.6 Entering a Field

Each time a manual (or automatic) field gets the focus in an attended module, an event is triggered:



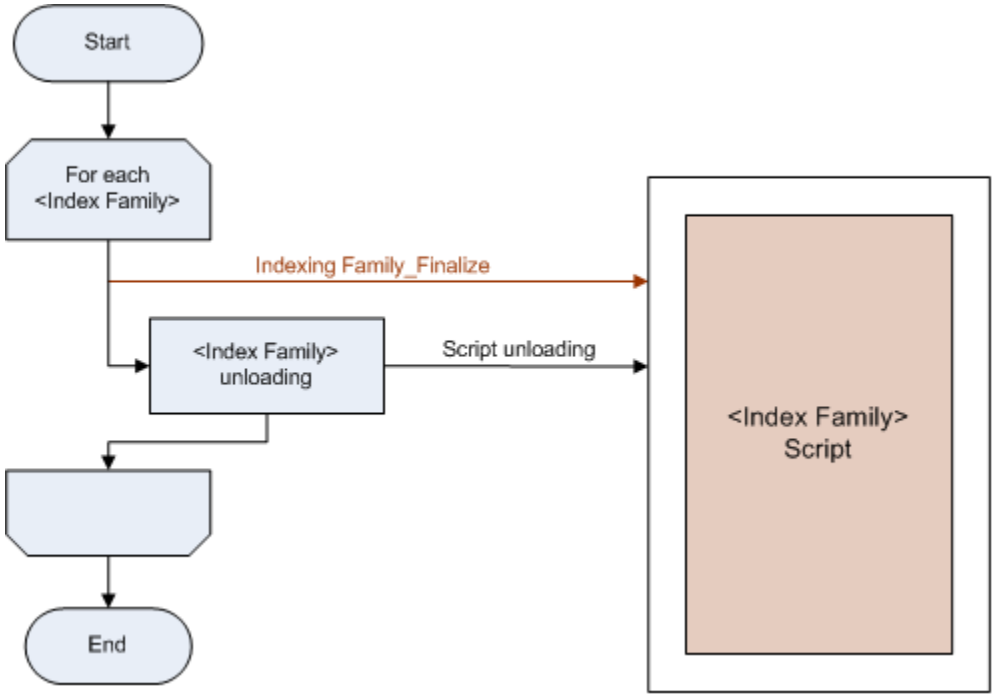
6.5.3.7 Loading Indexing Families

The following diagram shows the IndexingFamily_Initialize event triggered when each index family of a project is loaded.



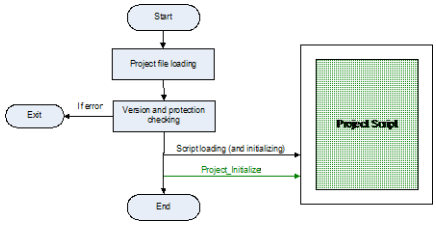
6.5.3.8 Unloading Indexing Families

The following diagram shows the IndexingFamily_Initialize event triggered when each index family of a project is unloaded.



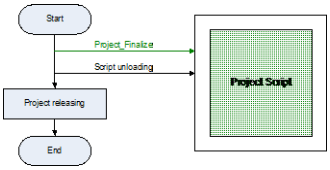
6.5.3.9 Project Loading

The project loading triggers only one event: Project_Initialize.



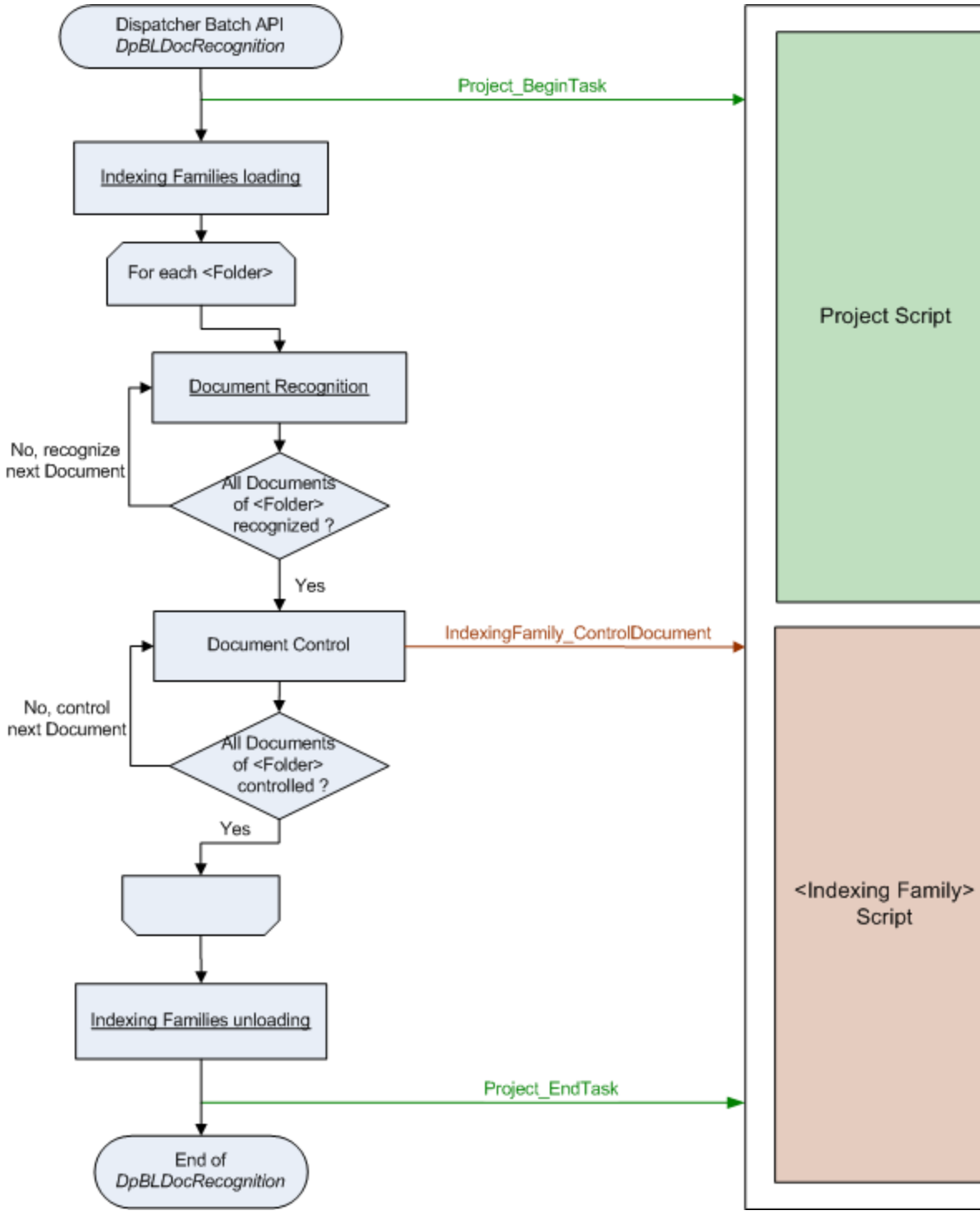
6.5.3.10 Project Unloading

The project unloading triggers only one event: Project_Finalize.



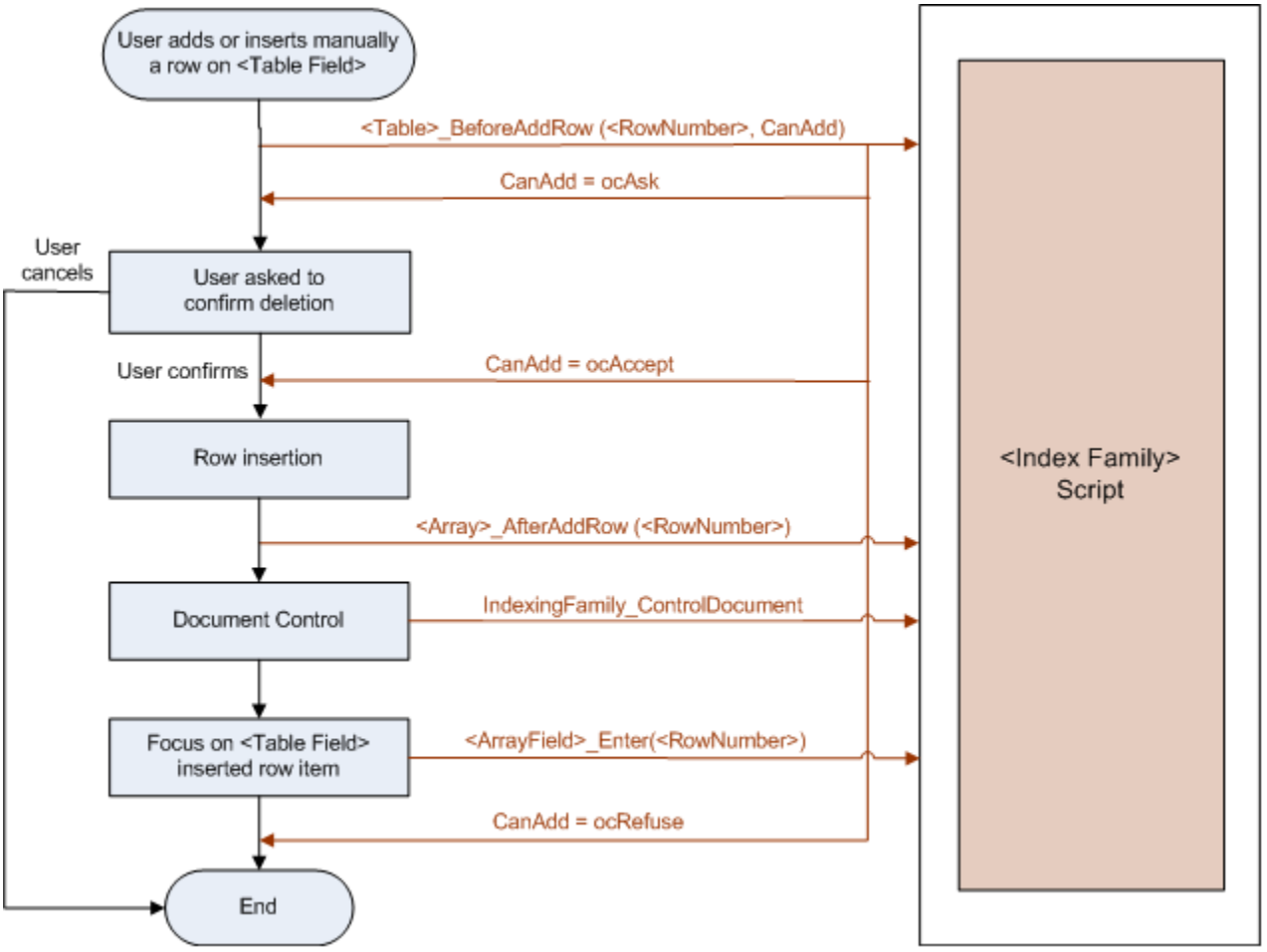
6.5.3.11 Recognition

The following diagram shows the event sequence of the recognition module:


Dispatcher Batch API
DpBLDocRecognition
Project_BeginTask
Indexing Families loading
For each <Folder>
Document Recognition
No, recognize next Document
All Documents of <Folder> recognized ?
Yes
Document Control
IndexingFamily_ControlDocument
No, control next Document
All Documents of <Folder> controlled ?
Yes
Indexing Families unloading
Project_EndTask
End of DpBLDocRecognition
Project Script
<Indexing Family> Script

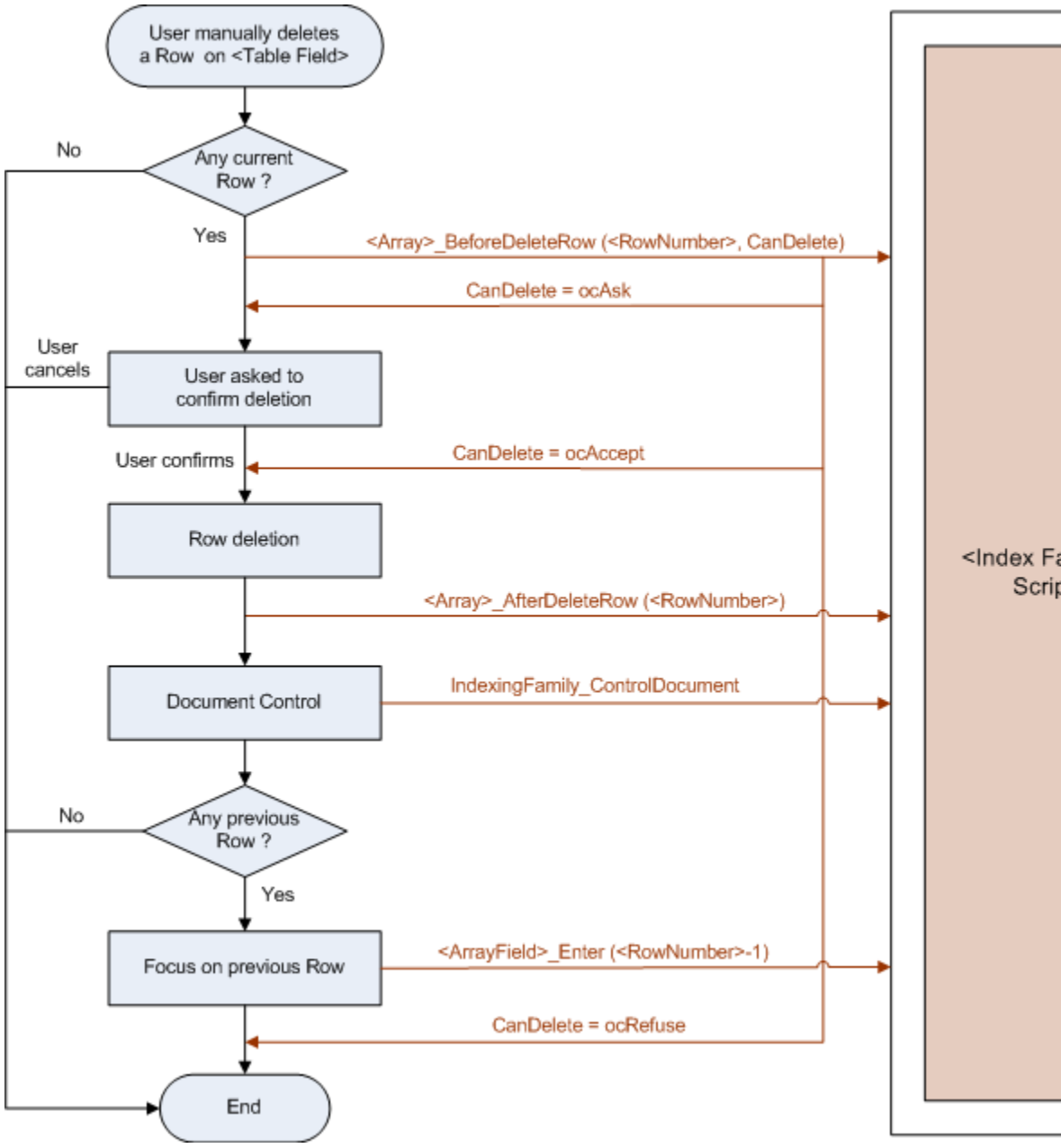
6.5.3.12 Row Addition

This diagram shows the events triggered when the user adds or inserts a row to a table field:



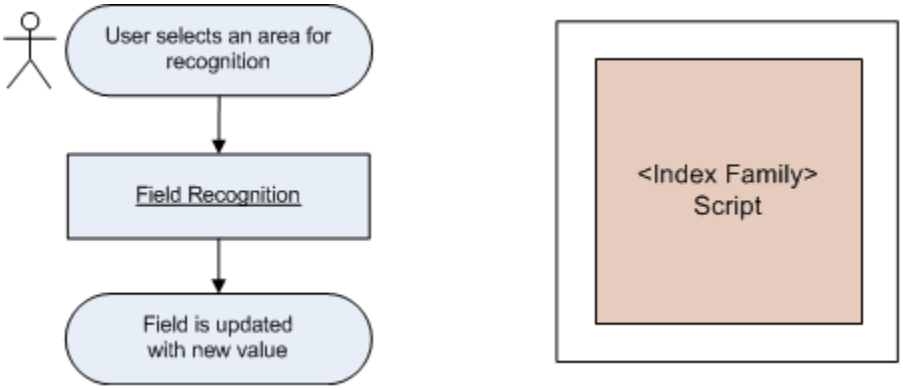
6.5.3.13 Row Deletion

This diagram shows the event sequence when the user deletes a row in a table field:


User manually deletes a Row on <Table Field>
Any current Row ?
No
Yes
<Array>_BeforeDeleteRow (<RowNumber>, CanDelete)
CanDelete = ocAsk
User cancels
User asked to confirm deletion
User confirms
CanDelete = ocAccept
Row deletion
<Array>_AfterDeleteRow (<RowNumber>)
Document Control
IndexingFamily_ControlDocument
Any previous Row ?
No
Yes
Focus on previous Row
<ArrayField>_Enter (<RowNumber>-1)
CanDelete = ocRefuse
End
<Index Fa
Scrip

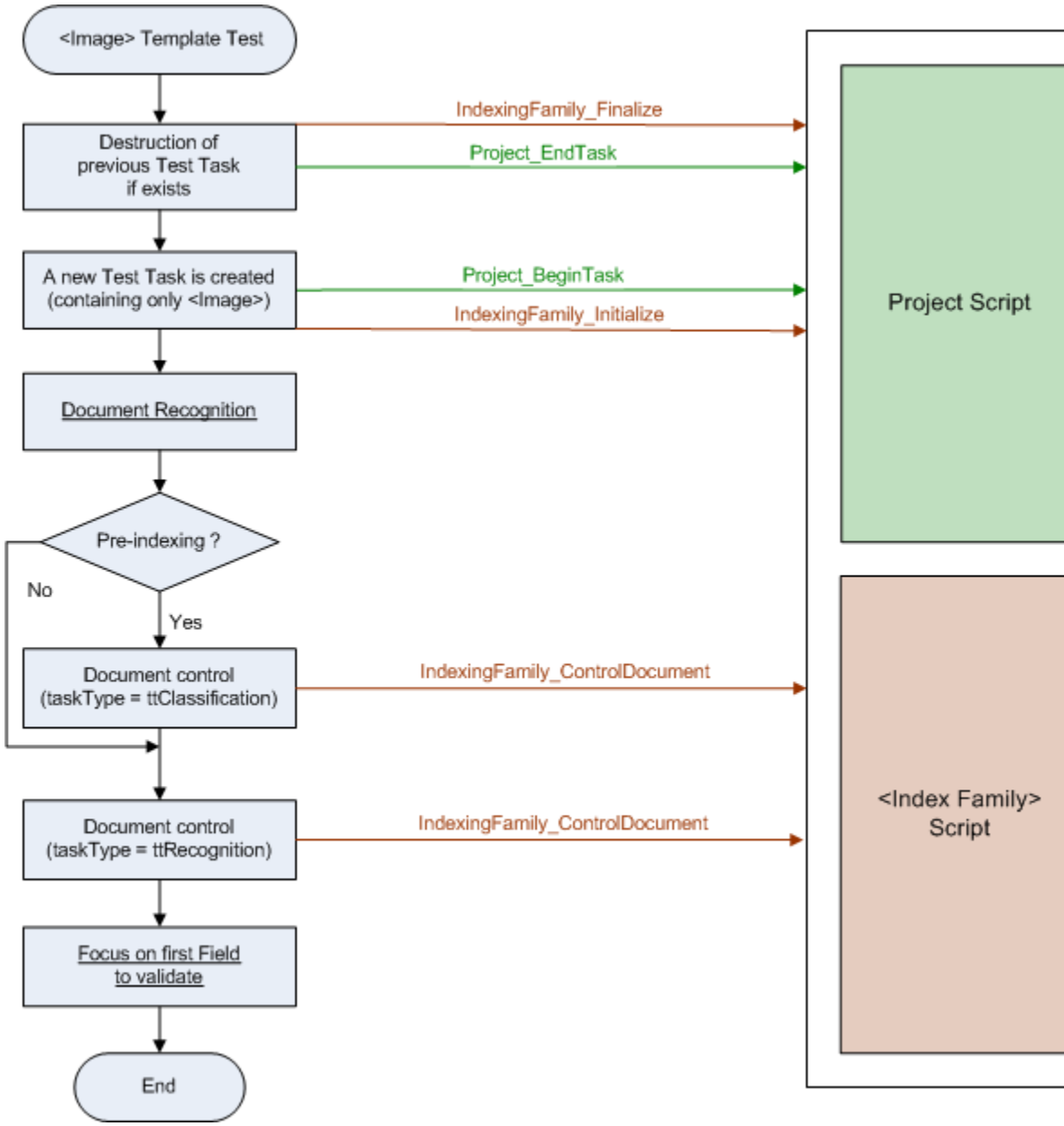
6.5.3.14 Rubber Band OCR

The events triggered during a rubber band *OCR* are the same as during field recognition:



6.5.3.15 Template Test: Single Image Test

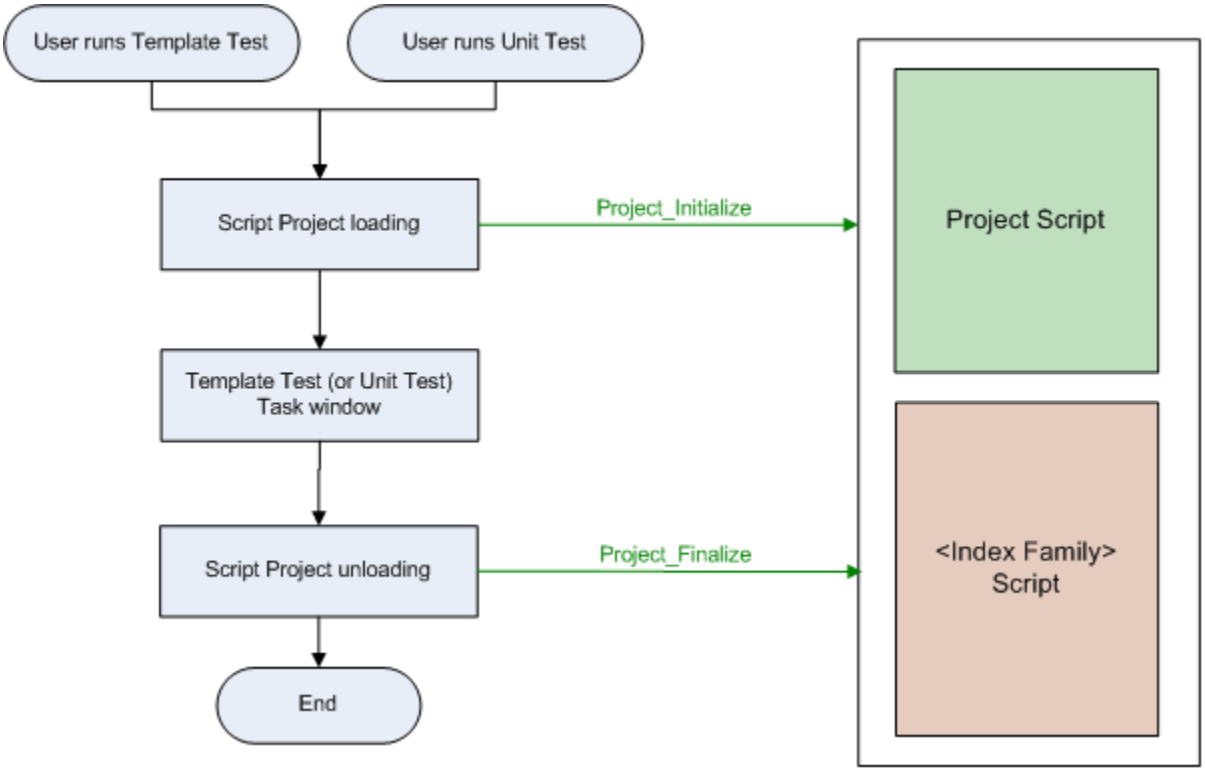
Each time a Template Test is done on one image, events are triggered as follows:



Note: Two consecutive controlDocument events can be triggered if the family contains pre-index fields. The first one is triggered with a ttClassification taskType value to simulate the controlDocument event in classification task. The second one, triggered with a ttRecognition taskType value, is related to the recognition task. If there are no pre-index fields, only the controlDocument event related to the recognition task is triggered.

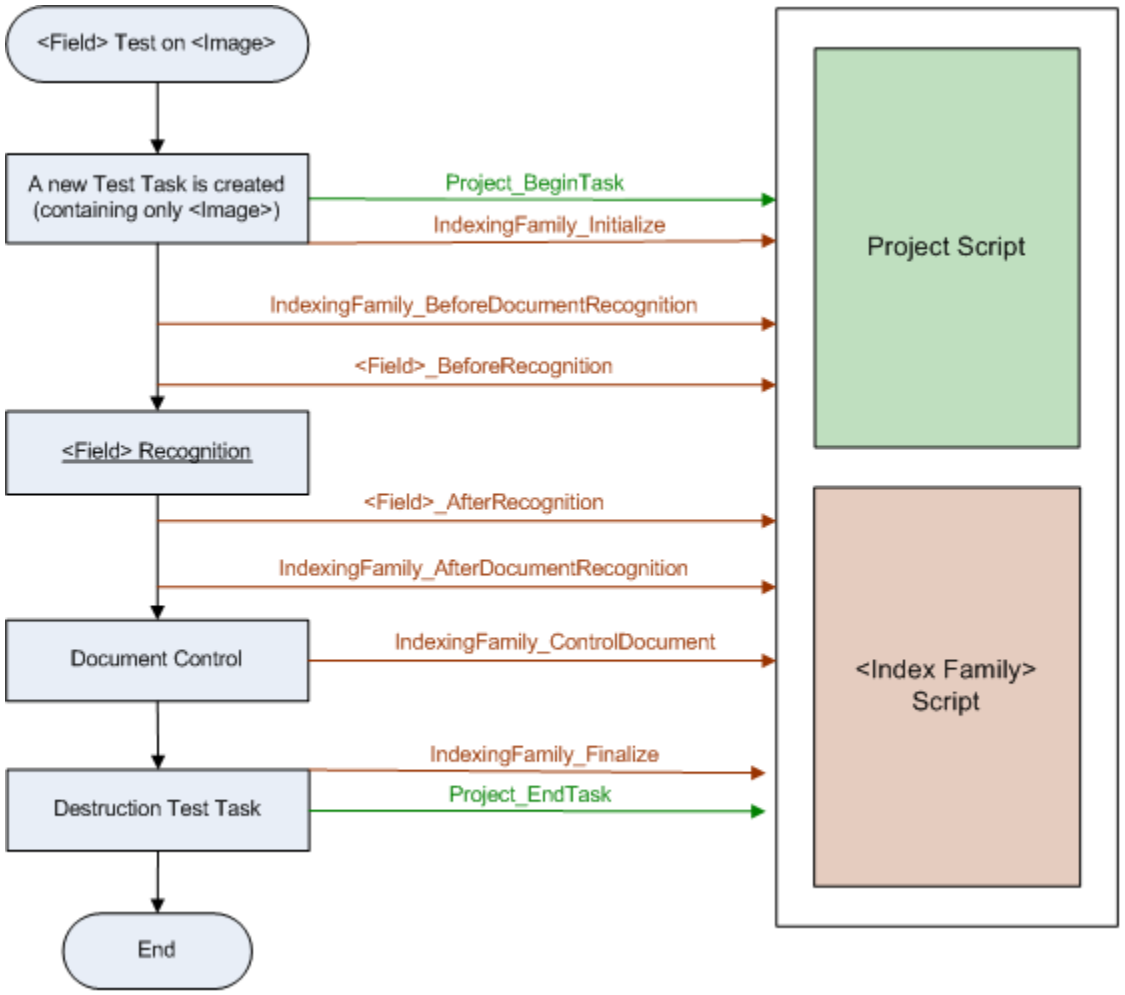
6.5.3.16 Template Test: Unit Test

This diagram shows the event sequence triggered when the Template Test or Unit Test are run and closed:



6.5.3.17 Unit Test: Single Image Test

Each time a Template Test is done on one image, events are triggered as follows:



6.6 Dispatcher Object Model

The Dispatcher Object Model contains all the internal objects such as a project, batch or document that are visible in the interface. Each of these objects has specific properties, methods and events. Methods enable performing actions on an object. Events constitute the Dispatcher Event Model.

The Dispatcher Event Model is used to create event handlers that are run when an event occurs. These event handlers then carry out certain actions on objects in the Dispatcher Object Model.

The Dispatcher Object Model is used by all Advanced Recognition modules. A matrix of Object Model elements is also available.

6.6.1 Object Model Overview

The main objects in the Dispatcher Object Model are:

- Task: Each task is attached to one specific Recognition project and contains the data from a batch containing one or more documents. Each task is processed by one specific Advanced Recognition module.
- Batch and Folder: When the Classification module starts running, the batch contains only one folder that contains all the documents.
- Document and Page
- Field Definition and Field Value: A field definition is a set of properties (Name, ID, etc.) that is defined in the template. A field value is the content of the field in the document; it changes with every document in production.
- Table

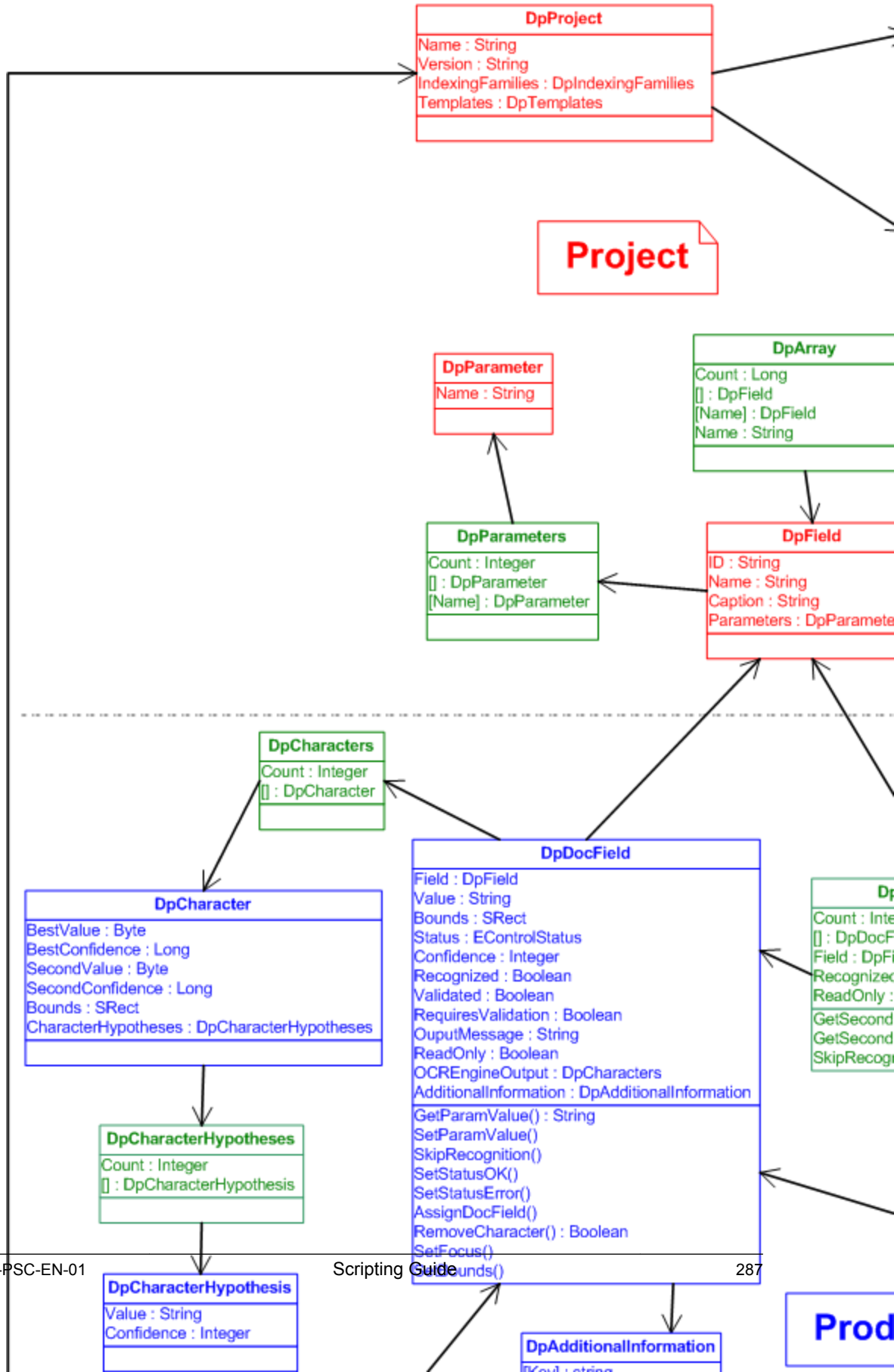
6.6.1.1 Execution Context

Certain actions on object properties or methods make sense only in a certain context; accessing these properties or methods in a different context can result in one of the following exceptions:

- Index family instances do not exist until they are loaded in the Project; attempting to access them returns the nothing keyword.
- During the indexing phase, attempting to modify the template on a classified document generates an exception. After Classification is finished, the template cannot be modified.

6.6.2 Object Model Schema

The following diagram shows all the object types defined in the Dispatcher Object Model and the interactions between them. Collections are in green. Objects in red are defined at project level (namely: templates, indexing families, fields) while objects in blue have values that change in production (namely: documents, batches, tasks, folders).


DpProject
Name : String
Version : String
IndexingFamilies : DpIndexingFamilies
Templates : DpTemplates
Project
DpParameter
Name : String
DpArray
Count : Long
[] : DpField
[Name] : DpField
Name : String
DpParameters
Count : Integer
[] : DpParameter
[Name] : DpParameter
DpField
ID : String
Name : String
Caption : String
Parameters : DpParamete
DpCharacters
Count : Integer
[] : DpCharacter
DpDocField
Field : DpField
Value : String
Bounds : SRect
Status : EControlStatus
Confidence : Integer
Recognized : Boolean
Validated : Boolean
RequiresValidation : Boolean
OuputMessage : String
ReadOnly : Boolean
OCREngineOutput : DpCharacters
AdditionalInformation : DpAdditionalInformation
GetParamValue() : String
SetParamValue()
SkipRecognition()
SetStatusOK()
SetStatusError()
AssignDocField()
RemoveCharacter() : Boolean
SetFocus()
DpCharacter
BestValue : Byte
BestConfidence : Long
SecondValue : Byte
SecondConfidence : Long
Bounds : SRect
CharacterHypotheses : DpCharacterHypotheses
DpCharacterHypotheses
Count : Integer
[] : DpCharacterHypothesis
DpCharacterHypothesis
Value : String
Confidence : Integer
DpAdditionalInformation
Prod

6.6.3 Object Types

This section details all the object types and collections defined in the Dispatcher Object Model as well as an object model schema that shows the interactions between them. A matrix of Object Model elements is also available.

The object types in this section can be used in a Recognition project:

6.6.3.1 DpAdditionalInformation

`DpAdditionalInformation` is a collection of strings which contains specific information about a `DpDocField`.

For example, it enables access to specific information about a field displaying a *US* check amount and which has been recognized with Parascript CheckPlus engine. This specific information includes values related to courtesy and legal amounts recognition (*CAR* and *LAR*).

Refer to the DpAdditionalInformation to learn more about the matrix of object model elements.

Accessors

- (Key): used to access a string by giving its key

The available keys can be used to access specific information about a `DpDocField` displaying a *US* check amount recognized with Parascript CheckPlus engine.

Table 6-135: Available Keys

Keys	Description
`"CheckplusUSCARValue"`	*CAR* value
`"CheckplusUSCARConfidence"`	*CAR* confidence value
`"CheckplusUSLARValue"`	*LAR* value
`"CheckplusUSLARConfidence"`	*LAR* confidence value
`"CheckplusUSCARLARMismatchConfidence"`	*CAR*/*LAR* mismatch confidence value
`"CheckplusUSLARCentsValue"`	Cents in legal amount
`"CheckplusUSLARCentsConfidence"`	Cents in legal amount confidence value
`"CheckplusUSLARDollarsValue"`	Dollars in legal amount
`"CheckplusUSLARDollarsConfidence"`	Dollars in legal amount confidence value
`"CheckplusUSCARCentsValue"`	Cents in courtesy amount
`"CheckplusUSCARCentsConfidence"`	Cents in courtesy amount confidence value
`"CheckplusUSCARDollarsValue"`	Dollars in courtesy amount

Keys	Description
"CheckplusUSCARDollarsConfidence"	Dollars in courtesy amount confidence value
"CheckplusUSCross-ValidatedCentsValue"	Cents in cross-validated amount
"CheckplusUSCross-ValidatedCentsConfidence"	Cents in cross-validated amount confidence value
"CheckplusUSCross-ValidatedDollarsValue"	Dollars in cross-validated amount
"CheckplusUSCross-ValidatedDollarsConfidence"	Dollars in cross-validated amount confidence value
"CheckplusUSCross-ValidatedValue"	Cross-validated amount value
"CheckplusUSCross-ValidatedConfidence"	Cross-validated amount confidence value

The following examples apply to recognition of a US check amount with Parascript CheckPlus engine.

Example: This example shows how to populate the current field with the *LAR* value if the *CAR* value confidence threshold is lower than the *LAR* value confidence threshold.

```
Private Sub US_Amount_AfterRecognition()
 Dim LAR As String
 Dim CARConfidence As Long
 Dim LARConfidence As Long
 LAR=CurrentField.AdditionalInformation("CheckplusUSLARValue")
 CARConfidence= Val(CurrentField.AdditionalInformation("CheckplusUSCARConfidence"))
 LARConfidence= Val(CurrentField.AdditionalInformation("CheckplusUSLARConfidence"))
 If CARConfidence<LARConfidence Then
   CurrentField.Value = LAR
  End if
End Sub
```

Example: This example shows how to detect fraudulent checks by setting a threshold to be compared with the *CAR*/*LAR* mismatch confidence value.

```
Private Sub US_Amount_AfterRecognition()
 Dim CARLARMismatchConfidence As Long
 CARLARMismatchConfidence =
Val(CurrentField.AdditionalInformation("CheckplusUSCARLARMismatchConfidence"))
 If CARLARMismatchConfidence>80 Then
   CurrentField.Value = "fraudulent"
 End If
End Sub
```

6.6.3.2 DpArray

`DpArray` is a collection of DpFields objects. To see the `DpArray` matrix tables, refer to DpArray.

Accessors

- Count: counts the number of `DpField` objects in this collection
- (): used to access a `DpField` object by giving its rank (first object takes rank 0)
- (Name): used to access a `DpField` object by giving its Name

Properties

- "Name" on page 324

Events

- "<Table>_BeforeAddRow" on page 266
- "<Table>_AfterAddRow" on page 267
- "<Table>_BeforeDeleteRow" on page 268
- "<Table>_AfterDeleteRow" on page 269

6.6.3.3 DpArrays

`DpArrays` is a collection of DpArray objects. To see the `DpArrays` matrix tables, refer to DpArrays.

Accessors

- Count: counts the number of `DpArray` objects in this collection
- (): used to access a `DpArray` object by giving its rank (first object takes rank 0)
- (Name): used to access a `DpArray` object by giving its Name

6.6.3.4 DpBatch

`DpBatch`:

- Is the object type for a batch, which is the highest level in the document structure hierarchy.
- Contains one or more DpFolders (a batch contains at least one folder).
- Cannot be edited by a script, that is folders cannot be inserted or deleted.

To see the `DpBatch` matrix tables, refer to DpBatch.

Properties

- "Name" on page 324

- "ID" on page 319
- "Folders" on page 322
- "ExternalValues" on page 320

Methods

None

Events

None

6.6.3.5 DpCharacter

DpCharacter is an object type that is used to manage field characters. The object:

- Returns character level information for each DpDocField value when zonal *OCR* is performed.
- Returns character level information for each DpDocField value when Free Form *OCR* is performed.
- Returns character level information for any supported language.

This object is only accessible from the Field_AfterRecognition event.

The properties BestValue, SecondValue, BestConfidence, and SecondConfidence have been deprecated. Although these properties remain in the *DOM* for backward compatibility, they are not Unicode compliant and will return inconsistent values if used with Asian language documents. New Confidence and CharacterHypotheses properties replace the deprecated properties.

To see the DpCharacter matrix tables, refer to DpCharacter.

Properties

- "Bounds" on page 309
- "CharacterHypotheses" on page 310
- Deprecated properties and replacements:
  - "BestConfidence" on page 308: Use "Confidence (DpCharacterHypothesis)" on page 311.
  - "BestValue" on page 308: Use "Value" on page 336 (CharacterHypothesis).
  - "SecondConfidence" on page 332: Use "Confidence (DpCharacterHypothesis)" on page 311.
  - "SecondValue" on page 332: Use "Value" on page 336 (CharacterHypothesis).

6.6.3.6 DpCharacters

DpCharacters is a collection of DpCharacter objects.

This object is only accessible from the Field_AfterRecognition event.

Accessors

- Count: counts and returns the number of DpCharacter objects in this collection.

  If OCREngineOutput.count = `0`, no character level information is returned, otherwise OCREngineOutput returns the number of characters found in the field.

- (): enables to access a DpCharacter object by its index.

To see the DpCharacters matrix tables, refer to DpCharacter.

6.6.3.7 DpCharacterHypotheses

Returns a set of character level information through the property OCREngineOutput in the DpDocField object. DpCharacterHypotheses is a collection of DpCharacterHypothesis objects.

This collection returns a list of all the character hypotheses returned by the recognition engine for recognized characters, whatever the document language.

This object is only accessible from the Field_AfterRecognition event.

Accessors

- Count: Counts and returns the number of DpCharacterHypothesis objects in this collection.

  If CharacterHypothesis.count = `0`, no character level information is returned. Otherwise CharacterHypothesis returns the number of hypothesis found for the field.

- (): enables to access a DpCharacterHypothesis object by its index.

To see the DpCharacterHypotheses matrix tables, refer to DpCharacterHypotheses.

6.6.3.8 DpCharacterHypothesis

Returns a character hypothesis in any supported language. Both Value and Confidence properties are retained.

Available from the `Field_AfterRecognition` event as the OCREngineOutput property of DpDocField.

Properties

- "Confidence (DpCharacterHypothesis)" on page 311: Returns the confidence value for the hypothesis of a character. Confidence values reflect the degree of

likelihood that the detected Value is correct and range from `0` to `100`. A confidence score of `100` means the character is fully recognized.

- "Value" on page 336: Returns the hypothesis value for a character.

Calls to deprecated properties must be replaced, as demonstrating in the following example.

Example: Access the first hypothesis value of the second character of a given field value

Previously: `OcrEngineOutput.item(1).BestValue`

Now:

```
OcrEngineOutput.item(1).CharacterHypotheses.item(0).value
==> Item(1 : second character
==> Item(0) : first hypothese
```

To see the DpCharacterHypothesis matrix tables, refer to DpCharacterHypothesis.

6.6.3.9 DpDocArray

`DpDocArray` is a collection of DpDocArrayField objects.

Accessors

- Count: counts the number of `DpDocArrayField` objects in this collection
- (): used to access a `DpDocArray` object by giving its rank (first object takes rank 0)
- (Name): used to access a `DpDocArray` object by giving its Name

Properties

- "CurrentRow" on page 317
- "RowCount" on page 331

Methods

- "AddRow" on page 133
- "DeleteRow" on page 134
- "IsParagraphHeader" on page 137
- "IsUserRow" on page 137

To see the `DpDocArray` matrix tables, refer to DpDocArray.

6.6.3.10 DpDocArrays

`DpDocArrays` is a collection of DpDocArray objects.

Accessors

- Count: counts and returns the number of `DpIndexingFamily` objects in this collection.
- (): used to access a `DpDocArray` object by giving its rank (first object takes rank 0)
- (Name): used to access a `DpDocArray` object by giving its Name

To see the `DpDocArrays` matrix tables, refer to DpDocArrays.

6.6.3.11 DpDocArrayField

`DpDocArrayField` is a collection of all row item values of the Table Field (DpDocField).

Accessors

- Count: counts the number of `DpDocField` objects in this collection
- (): used to access a `DpDocArrayField` object by giving its rank (first object takes rank 0)

Properties

- "Field" on page 320
- "ReadOnly" on page 328
- "Recognized" on page 329

Methods

- "GetSecondaryValue" on page 135
- "GetSecondaryValueCount" on page 136
- "SkipRecognition" on page 140

To see the `DpDocArrayField` matrix tables, refer to DpDocArrayField.

6.6.3.12 DpDocField

`DpDocField` is the object type that uses properties for returning index field values and the specific row item of Table Field through the DpDocField Value property.

Properties

- "AdditionalInformation" on page 307
- "Bounds" on page 309
- "Confidence (DpDocField)" on page 312
- "Field" on page 320
- "OCREngineOutput" on page 324
- "OutputMessage" on page 325
- "ReadOnly" on page 328
- "Recognized" on page 329
- "RequiresValidation" on page 329
- "Value" on page 336

Methods

- AssignDocField
- GetParamValue
- RemoveCharacter
- SetBounds
- SetFocus
- SetStatusOK
- SetStatusError
- SetParamValue
- SkipRecognition

Events

None

Related Topics

"DpDocField" on page 380

Confidence (DpDocField)

6.6.3.13 DpDocFields

`DpDocFields` is a collection of DpDocField objects.

Accessors

- Count: counts the number of `DpDocField` objects in this collection
- (): used to access a `DpDocField` object by giving its rank (first object takes rank 0)
- (Name): used to access a `DpDocField` object by giving its Name

To see the `DpDocFields` matrix tables, refer to DpDocFields.

6.6.3.14 DpDocument

`DpDocument`: Is the object type for documents and:

- Contains one or more pages
- Is always contained in a parent DpFolder
- Is the only access to document information such as template or field value.
- Returns the template properties for the five main template hypotheses computed in classification.

Properties

- "ID" on page 319
- "Pages" on page 326
- "Template" on page 334
- "TemplateProperties" on page 335
- "TemplateHypotheses" on page 335
- "IndexingFamily" on page 323
- "Folder" on page 322
- "RankInFolder" on page 328
- "Arrays" on page 307
- "Fields" on page 321
- "Status" on page 333
- "Classified" on page 310
- "Recognized" on page 329
- "Rejected" on page 331
- "ExternalValues" on page 320
- "RectifiedPaperOrientation" on page 330

- "RectifiedDuplexInversion" on page 330

Methods

- "GetPreviousDocument" on page 135
- "GetNextFolder" on page 136
- "SkipRecognition" on page 140

Events

None

The `DpDocument` matrix tables are presented in DpDocument.

6.6.3.15 DpDocuments

`DpDocuments` is a collection of DpDocument objects.

Accessors

- (): used to access a `DpDocument` object by giving its rank (First object takes rank 0)
- (ID): used to access a `DpDocument` object by giving its ID

To see the `DpDocuments` matrix tables, refer to DpDocuments.

6.6.3.16 DpDocTemplate

The `DpDocTemplate` Represents the classification properties of the template. Theconfidence value represents the template classification decision rate.

Properties

- "Template" on page 334
- "Confidence (DpDocTemplate)" on page 312
- "PaperOrientation" on page 326
- "PreClassificationRate" on page 327
- "DuplexInversion" on page 318

To see the `DpDocTemplate` matrix tables, refer to DpDocTemplate.

6.6.3.17 DpExternalValue

This property contains the number of the items in the collection.

Each external value can be accessed from its index. The n-th external value in the collection is accessed with the following code:

Example:

```
Dim extValue as DpExternalValue
extValue = document.ExternalValues(n)
```

The first item is always at index 0.

If the index is out of bounds, the returned value is the VBA null value.

Each external value can be accessed from its name. The external value which name is <name> is accessed with the following code:

Example:

```
Dim extValue as DpExternalValue
extValue = document.ExternalValues("<name>")
```

If the external value name does not exist, the returned value is the VBA "null" value.

Properties

- "Name" on page 324
- "Value" on page 336

To see the `DpExternalValue` matrix tables, refer to DpExternalValue.

6.6.3.18 DpExternalValues

`DpExternalValues` is a collection of DpExternalValue objects.

At execution, the `ExternalValues` collection contains the IA values that have been selected in the setup mode of the module.

- IA values of levels 0 (Page) and 7 (Batch) are available in the Classification module.

Accessors

- Count: counts and returns the number of `DpExternalValue` objects in this collection.
- (): used to access a `DpExternalValue` object by giving its rank (First object takes rank 0)
- (Name): used to access a `DpExternalValue` object by giving its Name.

To see the `DpExternalValues` matrix tables, refer to DpExternalValues.

6.6.3.19 DpField

`DpField`:

- Is the object type for Fields (index fields and table fields)
- Contains the collection of DpParameters but not the field value (the field value is contained in DpDocField)

Properties

- "ID" on page 319
- "Name" on page 324
- "Parameters" on page 327

Methods

None

Events

- "<Field>_BeforeRecognition" on page 264
- "<Field>_AfterRecognition" on page 265

To see the `DpField` matrix tables, refer to DpField.

6.6.3.20 DpFields

Accessors

- Count: counts and returns the number of `DpIndexingFamily` objects in this collection.
- (): used to access a `DpField` object by giving its rank (First object takes rank 0)
- (Name): used to access a `DpField` object by giving its Name

To see the `DpFields` matrix tables, refer to DpFields.

6.6.3.21 DpFolder

`DpFolder`: The object type for folders and:

- Contains DpDocument objects.
- Cannot be edited, that is, documents cannot be inserted or deleted.

Properties

- "DpDocuments" on page 297
- "DpExternalValues" on page 298

Methods

- "GetPreviousFolder" on page 136
- "GetNextFolder" on page 136

Events

None

To see the `DpFolder` matrix tables, refer to DpFolder.

6.6.3.22 DpFolders

`DpFolders` is a collection of DpFolder objects.

Accessors

- *`Count`*: counts and returns the number of `DpIndexingFamily` objects in this collection.
- (): used to access a `DpFolder` object by giving its rank (First object takes rank 0)

To see the `DpFolders` matrix tables, refer to DpFolders.

6.6.3.23 DpIndexingFamily

`DpIndexingFamily` is the object type for indexing families. It contains field definitions and parameters, but cannot be edited. It is also the main object to trigger the recognition events.

Properties

- "Name" on page 324
- "Fields" on page 321
- "Arrays" on page 307

Methods

None

Events

- "IndexingFamily_Initialize" on page 263
- "IndexingFamily_Finalize" on page 271
- "IndexingFamily_BeforeDocumentRecognition" on page 264
- "IndexingFamily_AfterDocumentRecognition" on page 265
- "IndexingFamily_ControlDocument" on page 269
- "IndexingFamily_EnterDocument" on page 266
- "IndexingFamily_ExitDocument" on page 271
- "IndexingFamily_DocumentValidated" on page 269
- "IndexingFamily_PressCustomHotKey" on page 270

To see the `DpIndexingFamily` matrix tables, refer to DpIndexingFamily.

6.6.3.24 DpIndexingFamilies

`DpIndexingFamilies` is a collection of DpIndexingFamily objects.

Accessors

- *`Count`*: counts and returns the number of `DpIndexingFamily` objects in this collection.
- (): used to access a `DpIndexingFamilies` object by giving its rank (First object takes rank 0)
- (*`Name`*): used to access a `DpIndexingFamily` object by giving its Name

To see the `DpIndexingFamilies` matrix tables, refer to DpIndexingFamilies.

6.6.3.25 DpLineItems

`DpLineItems` is a collection of strings. It is not a collection of Dispatcher Object Model objects, as other collections usually are.

Accessors

- *`Count`*: counts the number of strings in this collection.
- (): used to access a string value by giving its rank (the first string takes rank 0) or its name.

Example: The column with the position n in the relation column list can be accessed with either of the following codes:

```
columnValue = lineItems.Item(n)
columnValue = lineItems(n)
```

Example: Each collection item value can be accessed with the column name. The column named “Column” in the DFT can be accessed with either of the following codes:

```
columnValue = lineItems.Item(“Column”)
columnValue = lineItems(“Column”)
```

6.6.3.26 DpPage

A `DpPage` is the object type for the page image: each `DpPage` is associated to a *TIFF* or a JPEG file.

Properties

- “FilePath” on page 321

Methods

None

Events

None

To see the `DpPage` matrix tables, refer to DpPage.

6.6.3.27 DpPages

`DpPages` is a collection of DpPage objects.

Accessors

- (): used to access a `DpPage` object by giving its rank (First object takes rank 0)

To see the `DpPages` matrix tables, refer to DpPages.

6.6.3.28 DpParameter

`DpParameter` is the object type for Field parameters.

Properties

- “Name” on page 324

Methods

None

Events

None

To see the `DpParameter` matrix tables, refer to DpParameter.

6.6.3.29 DpParameters

`DpParameters` is a collection of DpParameter objects.

Accessors

- *`Count`*: counts and returns the number of `DpTemplate` objects in this collection.
- (): used to access a `DpParameter` object by giving its rank (First object takes rank 0)
- (*`Name`*): used to access a `DpParameter` object by giving its Name

To see the `DpParameters` matrix tables, refer to DpParameters.

6.6.3.30 DpProject

`DpProject` is the object type for a Recognition project. It gives access, in reading mode, to the list of Recognition templates and indexing families.

Properties

- "Name" on page 324
- "Version" on page 337
- "IndexingFamilies" on page 323
- "Templates" on page 336

Methods

None

Events

- "Project_Initialize" on page 260
- "Project_Finalize" on page 263
- "Project_BeginTask" on page 260
- "Project_EndTask" on page 262
- "Project_BeforeClassification" on page 261
- "Project_AfterClassification" on page 262
- "Project_BeforeDocumentClassification" on page 261
- "Project_AfterDocumentClassification" on page 261

To see the `DpProject` matrix tables, refer to DpProject.

6.6.3.31 DpTask

`DpTask` is the object type for Tasks. `DpTask` is the highest level object.

Properties

- "TaskType" on page 334
- "CurrentProject" on page 316
- "CurrentBatch" on page 314
- "CurrentFolder" on page 316
- "CurrentDocument" on page 314
- "CurrentField" on page 315
- "CurrentArrayField" on page 313
- "CurrentTask" on page 317

Methods

None

Events

None

To see the `DpTask` matrix tables, refer to DpTask.

6.6.3.32 DpTemplate

`DpTemplate` is the object type for a Recognition template; `DpTemplate` contains the properties given in the table below in read mode only (except for the parameter value).

Properties

- "Name" on page 324
- "ID" on page 319
- "Code" on page 311
- "IndexingFamily" on page 323
- "Separator" on page 333

Methods

None

Events

None

To see the `DpTemplate` matrix tables, refer to DpTemplate.

6.6.3.33 DpTemplates

`DpTemplates` is a collection of DpTemplate objects.

Accessors

- *`Count`*: counts and returns the number of `DpTemplate` objects in this collection.
- (): used to access a `DpTemplate` object by giving its rank (First object takes rank 0)
- (*`Name`*): used to access a `DpTemplate` object by giving its Name

Methods

- GetTemplateByID

To see the `DpTemplates` matrix tables, refer to DpTemplates.

6.6.3.34 DpTemplateHypotheses

This collection of DpDocTemplate objects contains a maximum of five `DpDocTemplate` objects.

Accessors

- *`Count`*: counts the number of `DpDocTemplate` objects in this collection
- (): used to access a `DpDocTemplate` object by giving its rank (first object takes rank 0)
- (*`Name`*): used to access a `DpDocTemplate` object by giving its Name

6.6.4 Enumerations

6.6.4.1 ETaskType

This enumeration defines the Advanced Recognition module in which the script is run: Classification, Template Test, and Unit Test. The possible values are:

- `ttUnknown`
- `ttClassification`

The value `ttUnknown` is used as a default value just after the creation of a task and it will be set only for the DpTask objects explicitly created in scripts (`ttUnknown` is also a default internal value for object; it is not available in script execution). The type applies to `DpTask` objects.

6.6.4.2 EControlStatus

This enumeration defines the possible result value of a control applied to a field value. The possible values are:

- `csOK`
- `csError`
- `csNone` (`csNone` until the control script is run).

The type applies to field values and documents.

6.6.4.3 EOperationControl

This enumeration defines the return value of several events for selecting whether an operation (such as a document reject) is accepted, refused or requires confirmation from the operator.

Possible values are:

- `ocAccept` (the operation is accepted and will be run),
- `ocRefuse` (the operation is not accepted and will not be run),
- `ocAsk` (a message window appears asking the operator to confirm the operation).

6.6.4.4 EScriptError

This enumeration defines the error codes of a script exception.

The only possible value is `seBadContextError`: it is the error code used to raise a bad context error in a script.

6.6.4.5 ESearchDirection

This enumeration defines an input parameter which specifies the search direction of items. For example, to search a secondary row value, you can specify a direction relative to the primary row value. The possible values are:

- `sdAbove` (to search above the item)
- `sdBelow` (to search below the item)
- `sdBoth` (to search above and below the item)

6.6.5 Object Properties

This section details all the object type properties defined in the Dispatcher Object Model.

The properties in this section can be used in the objects of a Recognition project:

6.6.5.1 AdditionalInformation

Collection of strings which contains specific information about a `DpDocField`. For example, it enables access to specific information about a field displaying a US check amount and which has been recognized with Parascript CheckPlus engine. This specific information includes values related to courtesy and legal amounts recognition (CAR and LAR).

It is only available in the `AfterRecognition` event in the Recognition module.

Type

DpAdditionalInfomation

R/W

R

Default Value

Empty

Used

DpDocField

6.6.5.2 Arrays

Collection of tables (also known as arrays). Contains only one array object.

Type

DpDocArrays

R/W

R

Default Value

Empty

Used

DpDocument

DpIndexingFamily

6.6.5.3 BestValue

Returns the best hypothesis value for a character.

This property is not Unicode-compliant and has been deprecated. It remains in the *DOM* for backward-compatibility only. It is highly recommended that this property be replaced by the new property "Value" on page 336 (DpCharacterHypothesis).

Call to the BestValue property must be replaced by `CharacterHypotheses(0).Value`.

Type

Byte

R/W

R

Default Value

Empty

Used

DpCharacter

6.6.5.4 BestConfidence

Returns confidence rate for corresponding `DpCharacter::<BestValue>`. The returned value ranges from 0 to 100 when zonal *OCR* or Free Form *OCR* is performed.

This property is not Unicode-compliant and has been deprecated. It remains in the *DOM* for backward-compatibility only. It is highly recommended that this property be replaced by the new property "Confidence (DpCharacterHypothesis)" on page 311.

Calls to theBestConfidence property must be replaced by `CharacterHypotheses(0).Confidence`

Type

Long

R/W

R

Default Value

0

Used

DpCharacter

6.6.5.5 Bounds

Returns the bounds of a character in the field, or the field in the page respectively.

- `DpCharacter`: Returns the bounds of a character in the field. The bounds unit is measured in pixels and area coordinates are defined in relation to field area, hence the coordinates (0,0) define the left top corner of field area.
- `DpDocField`: Returns the bounds of a field in the page. The bounds unit is measured in pixels and area coordinates are defined in relation to page area, hence the coordinates (0,0) define the left top corner of page area.

Type

SRect. This object type enables the handling of coordinate information for the rectangles that are used by Dispatcher Object Model objects

R/W

R

Default Value

(0, 0, 0, 0)

Used

DpDocField

DpCharacter

6.6.5.6 CharacterHypotheses

This property will return all character hypotheses. It is a Unicode-compliant property that replaces the properties "BestValue" on page 308, "SecondValue" on page 332, "BestConfidence" on page 308, and "SecondConfidence" on page 332.

Though these deprecated properties are still available in the *DOM* for backward compatibility, it is highly recommended that *VBA* code be updated to replace the deprecated properties with CharacterHypotheses.

The CharacterHypotheses property is only available with the *<Field>*`_AfterRecognition` event as an OCREngineOutput property of "DpDocField" on page 295.

Type

Boolean

R/W

R

Default Value

Empty

Used

"DpCharacter" on page 291

"DpCharacterHypothesis" on page 292

6.6.5.7 Classified

True if the *`Document`* has an associated `Template` (Template property), False otherwise.

Type

Boolean

R/W

R

Default Value

False

Used

DpDocument

6.6.5.8 Code

Template code as defined in the project.

Type

String

R/W

R

Default Value

“”

Used

DpDocTemplate

6.6.5.9 Confidence (DpCharacterHypothesis)

Return a confidence rate for the corresponding `DpCharacterHypothesis::<Value>`.

`DpCharacterHypothesis::<Confidence>` returns a value between `0` and `100`, whichever engine is used for recognition. A confidence score of `100` means that the character is entirely recognized.

Type

Integer

R/W

R

Default Value

0

Used

DpCharacterHypothesis

6.6.5.10 Confidence (DpDocTemplate)

Represents the template classification decision rate. This value has a possible range between 0 and 100. A 100 value means that the document is fully classified.

The `DpDocTemplate::`*`Confidence`* value is 100 when:

- The document is classified with an HPA template
- The document is classified with keyword rules
- The document is assigned to a handwritten template
- The document is assigned to a default template
- The `DpDocument::`*`Template`* is assigned by scripting

The `DpDocTemplate::`*`Confidence`* value is between 0 and 100 when:

- The document is classified with a standard template
- The document is classified with a text matching template
- `DpDocument::`*`TemplateProperties`* is assigned by scripting

Type

Long

R/W

R

Default Value

0

6.6.5.11 Confidence (DpDocField)

Represents the field confidence rate. This property is only set by the following recognition engines during the field recognition step:

- Barcode Recognition (only if the **Maximum barcode per image** option is set to 1)
- Barcode 39 Recognition
- Modification Detection
- Check Reading

This value has a possible range between 0 and 100. A 100 value means that the field is fully recognized.

DpDocField.Confidence is set to –1 if the field returns character level information, otherwise it returns a value between 0 and 100.

Type

Integer

R/W

Read only

Default Value

-1

6.6.5.12 CurrentArrayField

This property is used to process the current table field of the document. The *`CurrentArrayField`* property is only available if the current field is a table field, and with the following events:

- `<Field>_BeforeRecognition`
- `<Field>_AfterRecognition`
- `IndexingFamily_PressCustomHotKey`

The *`CurrentArrayField`* property is set to `Nothing` everywhere except in the events listed above.

If the current field is an index field, *`CurrentArrayField`* is set to Nothing.

Type

DpTask

R/W

R

Default Value

Nothing

Used

DpTask

6.6.5.13 CurrentBatch

Current batch associated to the task.

Set to Nothing in the following events: `Project_Intialize`, `Project_Finalize`, `Project_BeginTask`, `Project_EndTask`, `IndexingFamily_Initialize`, `IndexingFamily_Finalize`.

Type

DpBatch

R/W

R

Default Value

Nothing

Used

DpTask

6.6.5.14 CurrentDocument

Current document processed.

Set to Nothing in the following events:

- `Project_Initialize`
- `Project_Finalize`
- `Project_BeginTask`
- `Project_EndTask`
- `IndexingFamily_Initialize`
- `IndexingFamily_Finalize`
- `Project_BeforeClassification`
- `Project_AfterClassification`

Type

DpDocument

R/W

R

Default Value

Nothing

Used

DpTask

6.6.5.15 CurrentField

This property is used to process the current index field or the current item of a table field.

Current Index Field

- If *CurrentField* is an index field, it is defined in the following events:
  - `<Field>_BeforeRecognition`
  - `<Field>_AfterRecognition`
  - `IndexingFamily_PressCustomHotKey`

  For all other events, the *CurrentField* property is set to Nothing.

Current Item of a Table Field

- If the current field is a table field, *CurrentField* is set to the item value of the current row. It is defined in the following events:
  - `IndexingFamily_PressCustomHotKey`

  For all other events, the *CurrentField* property is set to Nothing.

Note: With table fields (also known as array fields), the *CurrentArrayField* variable must be used for the events `<Field>_BeforeRecognition` and `<Field>_AfterRecognition`.

Type

DpDocField

R/W

R

Default Value

Nothing

Used

DpTask

6.6.5.16 CurrentFolder

Current folder processed.

Set to Nothing in the following events:

- `Project_Intialize`
- `Project_Finalize`
- `Project_BeginTask`
- `Project_EndTask`
- `IndexingFamily_Initialize`
- `IndexingFamily_Finalize`

Type

DpFolder

R/W

R

Default Value

Nothing

Used

DpTask

6.6.5.17 CurrentProject

Current Batch associated to the task.

Set to Nothing in the following events:

- `Project_Intialize`
- `Project_Finalize`
- `Project_BeginTask`
- `Project_EndTask`
- `IndexingFamily_Initialize`
- `IndexingFamily_Finalize`

Type

DpProject

R/W

R

Default Value

Nothing

Used

DpTask

6.6.5.18 CurrentRow

Number of current row, if a Table Field has the focus, otherwise, set to -1. The first row has number 0

Type

Integer

R/W

R

Default Value

-1

Used

DpTask

6.6.5.19 CurrentTask

Returns the task object.

Type

DpTask

R/W

R

Default Value

Self

Used

DpTask

6.6.5.20 Documents

Collection of all the documents of the folder. Set to Empty by default.

- Count: counts the number of `DpDocTemplate` objects in this collection
- (): used to access a `DpDocTemplate` object by giving its rank (first object takes rank 0)
- (Name): used to access a `DpDocTemplate` object by giving its Name

Type

DpDocuments

R/W

R

Default Value

Empty

Used

DpFolder

6.6.5.21 DuplexInversion

Indicates whether or not a duplex inversion has been applied on a document.

The `DpDocTemplate::`*`DuplexInversion`* value is true or false when:

- The document is classified with a text matching template
- The document is classified with a standard template
- The document is classified using an HPA template

The `DpDocTemplate::`*`DuplexInversion`* value is false when:

- The document is classified using keyword rules
- The document is assigned to a handwritten template
- The document is assigned to a default template
- The `DpDocument::`*`Template`* is assigned by scripting

Type

Boolean

R/W

R

Default Value

False

Used

DpDocTemplate

6.6.5.22 ID

Used to give a unique identifier to a field, document, batch or template within the project.

Type

String

R/W

R

Default Value

“”

Used

DpBatch: Batch ID (used as a parameter in some *DFL* functions).

DpDocument- Document ID. Used as a parameter in some *DFL* functions.

DpDocField: Field ID as defined in the project.

DpTemplate: Template ID as defined in the project.

6.6.5.23 ExternalValues

In the *DIA* context, at execution, the *`ExternalValues`* collection contains IA values that have been selected in the *DIA* module in setup mode.

`ExternalValues` never provokes a bad context error and is never null.

Type

DpExternalValues

R/W

R

Default Value

Empty

Used

DpBatch

DpDocument

DpFolder

6.6.5.24 Field

Link to the field definition object.

Type

DpField

R/W

R

Default Value

Nothing

Used

DpDocField

6.6.5.25 Fields

Collection of the index field values of the document or index family.

Type

DpDocField (`DpDocument`)

DpFields (`DpIndexingFamily`)

R/W

R

Default Value

Empty

Used

DpDocument

DpIndexingFamily

6.6.5.26 FilePath

Full path of the page-image file.

Type

String

R/W

R

Default Value

“”

Used

DpPage

6.6.5.27 Folder

Folder containing the current document.

Type

DpFolder

R/W

R

Default Value

Nothing

Used

DpDocument

6.6.5.28 Folders

Collection of all the folders of the batch.

Type

DpFolders

R/W

R

Default Value

Empty

Used

DpBatch

6.6.5.29 IndexingFamily

Returns the associated index family as defined in the project. Set to Nothing if there is no associated index family.

Type

`DpIndexFamily` (`DpDocument`)

`DpIndexingFamily` (`DpTemplate`)

R/W

R

Default Value

Nothing

Used

DpDocument

DpTemplate

6.6.5.30 IndexingFamilies

Collection of all the indexing families defined in the project. Defined and loaded in the project.

Type

DpIndexingFamilies

R/W

R

Default Value

Empty

Used

DpProject

6.6.5.31 Name

The name of the object (array, batch, external value, field, indexing family, parameter, project or template) as it is defined in the project.

Type

String

R/W

R

Default Value

“”

Used

DpArray

DpBatch

DpExternalValue

DpField

DpIndexingFamily

DpParameter

DpProject

DpTemplate

6.6.5.32 OCREngineOutput

Available for an index field, a table field, a rubber band field, an index field or a table field in the free form engine.

Returns an error when accessing `DpDocField:`*`OCREngineOutput`* somewhere else other than in the *`<Field>`*`_AfterRecognition` event.

Batch level functions return DPBL_ERROR_EXCEPTION status when accessing `DpDocField::`*`OCREngineOutput`* somewhere else other than in the *`<Field>`*`_AfterRecognition` event.

Page level functions return DPPL_ERROR_EXCEPTION status when accessing `DpDocField::`*`OCREngineOutput`* somewhere else other than in the `<Field>_AfterRecognition` event.

`OCREngineOutput` is not modified when `DpDocField::`*`Value`* is changed.

`DpDocField::`*`OCREngineOutput`* and `DpDocField::`*`Value`* data are not synchronized.

Type

DpCharacters

R/W

R

Default Value

Empty

Used

DpDocField

6.6.5.33 OutputMessage

Message that appears when the field or table field takes the focus in the **Template Test** window in Recognition Designer. This message appears for fields that require confirmation by the user or fields in error status.

Type

String

R/W

R/W

Default Value

“”

Used

DpDocField

6.6.5.34 Pages

Collection of all the pages of the document (usually only one or two).

Type

DpPages

R/W

R

Default Value

Empty

Used

DpDocument

6.6.5.35 PaperOrientation

Contains image rotation information. Possible values are:

- *poNormal*
- *poRotation90*
- *poRotation180*
- *poRotation270*

The DpDocTemplate*, PaperOrientation* value is between *poNormal* and *poRotation270* when:

- The document is classified using a standard template
- DpDocument::*TemplateProperties* is assigned by scripting
- The document is classified with a text matching template
- The document is classified using an *HPA* template

The DpDocTemplate::*PaperOrientation* value is *poNormal* when:

- DpDocument::*Template* is assigned by scripting

Type

EPaperOrientation

R/W

R

Default Value

poNormal

Used

DpDocTemplate

6.6.5.36 Parameters

Returns the collection of parameters if any parameter has been defined in the project; set to Empty otherwise.

Type

DpParameter

R/W

R

Default Value

Empty

Used

DpField

6.6.5.37 PreClassificationRate

Represents the template pre-classification rate with a range between 0 and 100.

The `DpDocTemplate` *`PreClassificationRate`* value is between 0 and 100 when:

- The document is classified using a standard template.
- The document is classified using an *HPA* template.
- `DpDocument`: *`TemplateProperties`* is assigned by scripting.
- The document is classified with a text matching template.

The `DpDocTemplate`:*`PreClassificationRate`* value is 100 when:

- The document is classified using keyword rules.
- The document is assigned to a handwritten template.
- The document is assigned to a default template.
- `DpDocument`:*`Template`* is assigned by scripting.

Type

Long

R/W

R

Default Value

0

Used

DpDocTemplate

6.6.5.38 RankInFolder

Rank of the current document in the folder. Always defined. First document takes rank 0.

Type

Long

Default Value

0

Used

DpDocument

6.6.5.39 ReadOnly

The value remains modifiable by script all the time.

Trying to modify this value for a `DpDocField` that belongs to a `DpDocArrayField` generates a bad context error.

Type

Boolean

R/W

R/W

Default Value

False

Used

DpDocField

6.6.5.40 Recognized

Indicates that recognition has been done or not on a field or document.

Type

Boolean

R/W

R

Default Value

True for `DpDocField`

False for `DpDocument`

Used

DpDocField

DpDocument

6.6.5.41 RequiresValidation

The field value requires a user validation.

Automatically cleared after a user validation on the field.

Set by default with the `Always confirm` value for the template field or index family field initially set in Recognition Designer.

Type

Boolean

R/W

R/W

Default Value

False

Used

DpDocField

6.6.5.42 RectifiedDuplexInversion

Indicates whether or not a duplex inversion has been applied on the document.

Set to false before the `Project_AfterClassification` event is called.

If a duplex inversion has been applied automatically to the document, *`RectifiedDuplexInversion`* property is set to true in the `Project_AfterClassification` event call and from thereon after.

Type

Boolean

R/W

R

Default Value

False

Used

DpDocument

6.6.5.43 RectifiedPaperOrientation

Indicates what rotation has been applied on the document. The default value indicates that no rotation has been applied on a document.

Set to *`poNormal`* before the `Project_AfterClassification` event is called.

If a rotation has been applied automatically or manually to the document, *`RectifiedPaperOrientation`* is set to a value other than *`poNormal`* in the `Project_AfterClassification` event call and afterwards.

`RectifiedPaperOrientation` is available according to the following access matrix.

Type

EPaperOrientation

R/W

R

Default Value

poNormal

Used

DpDocument

6.6.5.44 Rejected

Indicates that recognition has been done on a document.

Type

Boolean

R/W

R

Default Value

False

Used

DpDocument

6.6.5.45 RowCount

Primary row count used to count the number of `DpDocArrayField` objects in a collection.

Type

Long

R/W

R

Default Value

0

Used

DpDocArray

6.6.5.46 SecondConfidence

Returns a confidence rate for the corresponding `DpCharacter:`*`SecondValue`*. The returned value ranges from 0 to 100 when zonal *OCR* or Free Form *OCR* is performed.

This property is not Unicode-compliant and has been deprecated. It remains in the *DOM* for backward-compatibility only. It is highly recommended that this property be replaced by the new property "Confidence (DpCharacterHypothesis)" on page 311.

Calls to the SecondConfidence property must be replaced by `CharacterHypotheses(0).Confidence`.

Type

Long

R/W

R

Default Value

0 if no second value

Used

DpCharacter

6.6.5.47 SecondValue

Returns the second hypothesis value for a character.

This property is not Unicode-compliant and has been deprecated. It remains in the *DOM* for backward-compatibility only. It is highly recommended that this property be replaced by the new property "Value" on page 336 (DpCharacterHypothesis).

Calls to the SecondValue property must be replaced by `CharacterHypotheses(1).Value`.

Type

Byte

R/W

R

Default Value

Chr(0) if no second value

Used

DpCharacter

6.6.5.48 Separator

Template has been defined in the project as a separator or not.

Type

Boolean

R/W

R

Default Value

False

Used

DpTemplate

6.6.5.49 Status

Status of the document:

- *csOK*: If all the fields in the document have been recognized and they do not require validation or if the entire document is rejected.
- *csError*: If the document contains at least one field that is in error or at least one field that is correct but requires validation.
- *csNone*: If the document has not been classified yet.

Status of the field:

- *csOK*: If the field has been recognized and does not require validation.
- *csError*: If the field is in error or if the field is correct but requires validation.
- *csNone*: If the field has not been processed yet.

Type

EControlStatus

R/W

R

Default Value

csNone

Used

DpDocField

DpDocument

6.6.5.50 TaskType

Task type (ttUnknown, ttClassification, or ttRecognition.

Type

ETaskType

R/W

R

Default Value

ttUnknown

Used

DpTask

6.6.5.51 Template

If the document has been classified, returns the associated project template, otherwise, set to Nothing.

Used mostly in classification-level events to automatically classify the document by script code. Attempting to write to events other than `Project_BeforeClassification`, `Project_AfterClassification`, `Project_BeforeDocumentClassification` and `Project_AfterDocumentClassification` generates an exception.

The Template property is automatically updated when the *`TemplateProperties`* property is modified.

Type

`DpTemplate`

R/W

R/W

Default Value

Nothing

Used

DpDocument

DpDocTemplate

6.6.5.52 TemplateHypotheses

Collection of the five main template classification hypotheses for the document. Is set to Empty if no classification hypothesis was found.

Is available once the `Project_AfterDocumentClassification` event has been called. Set to default before this event is called.

Type

DpTemplateHypotheses

R/W

R

Default Value

Empty

Used

DpDocument

6.6.5.53 TemplateProperties

Contains the current classification template properties. This property is available after classification has been done on the document. There are two types of classification: by script or automatic.

`TemplateProperties` must be set with a `DpDocTemplate` value that belongs to the same *`TemplateHypotheses`* collection for the document.

`TemplateProperties` will not be modified if the `DpDocTemplate` value that is affected does not belong to the same *`TemplateHypotheses`* collection for the document: no script error raised.

`TemplateProperties` is automatically updated when the Template property is modified: Template property is set to the Template property value.

`TemplateProperties` property is set to Nothing before a classification has been done on the document.

Type

DpDocTemplate

R/W

R

Default Value

Nothing

Used

DpDocument

6.6.5.54 Templates

Collection of all the templates defined in the project.

Type

DpTemplates

R/W

R

Default Value

Empty

Used

DpProject

6.6.5.55 Value

Returns the value of a field, the row item value in a table field, the hypothesis value for a character, or an external value. Value can contain any supported character.

Type

String

R/W

- R/W: For the value of a field, or row item value of a table field.
- R: For the hypothesis value of a character.

Default Value

“”

Used

- DpDocField
- DpCharacterHypothesis.
- DpExternalValue

6.6.5.56 Version

Project version (composed of major and minor version number, for example “2.1”.

Type

String

R/W

R

Default Value

“”

Used

DpProject

6.7 Matrix Tables

This section contains tables that enable to visualize the different objects, events and properties and the relationships that link them together.

6.7.1 Matrix of Components

Table 6-136: Available Events by Module

Dispatcher component/module	Available events
Recognition Designer	• Project_Initialize (in template and field test only) • Project_Finalize (in template and field test only) • Project_BeginTask (in template and field test only) • Project_EndTask (in template and field test only) • IndexingFamily_Initialize (in template and field test only) • IndexingFamily_Finalize (in template and field test only) • IndexingFamily_BeforeDocumentRecognition (in template and field test only) • IndexingFamily_AfterDocumentRecognition (in template and field test only) • <Field>_BeforeRecognition (in template and field test only) • <Field>_AfterRecognition (in template and field test only) • IndexingFamily_ControlDocument (in template and field test only) • IndexingFamily_EnterDocument (in template test only) • IndexingFamily_DocumentValidated (in template test only) • IndexingFamily_ExitDocument (in template test only) • <Field>_BeforeAddRow (in template test only) • <Field>_AfterAddRow (in template test only) • <Table>_BeforeDeleteRow (in template test only) • <Table>_AfterDeleteRow (in template test only) • IndexingFamily_PressCustomHotKey (in template test only)

Dispatcher component/module	Available events
Classification	• Project_Initialize • Project_Finalize • Project_BeginTask • Project_EndTask • Project_BeforeClassification • Project_AfterClassification • Project_BeforeDocumentClassification • Project_AfterDocumentClassification • IndexingFamily_Initialize (if pre-indexing is used) • IndexingFamily_Finalize (if pre-indexing is used) • IndexingFamily_BeforeDocumentRecognition (if pre-indexing is used) • IndexingFamily_AfterDocumentRecognition (if pre-indexing is used) • <Field>_BeforeRecognition (if pre-indexing is used) • <Field>_AfterRecognition (if pre-indexing is used) • IndexingFamily_ControlDocument (if pre-indexing is used)

6.7.2 Matrix of Events

Table 6-137: Matrix of Events and Related Objects

Event	Parameters	Used in object ...
Project_Initialize	None	DpProject
Project_BeginTask	None	DpProject
Project_BeforeClassification	None	DpProject
Project_BeforeDocumentClassification	None	DpProject
Project_AfterDocumentClassification	None	DpProject
Project_AfterClassification	None	DpProject
Project_EndTask	None	DpProject
Project_Finalize	None	DpProject
IndexingFamily_Initialize	None	DpIndexingFamily

Event	Parameters	Used in object ...
IndexingFamily_BeforeDocumentRecognition	None	DpIndexingFamily
<Field>_BeforeRecognition	None	DpField
<Field>_AfterRecognition	None	DpField
IndexingFamily_AfterDocumentRecognition	None	DpIndexingFamily
IndexingFamily_EnterDocument	None	DpIndexingFamily
<Table>_BeforeAddRow	*CurrentRow* *CanAdd*	DpArray
<Table>_AfterAddRow	*CurrentRow*	DpArray
<Table>_BeforeDeleteRow	*CurrentRow* *CanDelete*	DpArray
<Table>_BeforeDeleteRow	*CurrentRow*	DpArray
IndexingFamily_DocumentValidated	None	DpIndexingFamily
IndexingFamily_ControlDocument	None	DpIndexingFamily
IndexingFamily_PressCustomHotKey	*KeyCode* *Shift* *Alt* *Ctrl* *Handled* *CurrentRow*	DpIndexingFamily
IndexingFamily_ExitDocument	None	DpIndexingFamily
IndexingFamily_Finalize	None	DpIndexingFamily

6.7.3 Matrix of Event Parameters

Refer to the Dispatcher Event Model and related parameters.

Table 6-138: Matrix of Event Parameters

Parameter	Used in event
Alt	IndexingFamily_PressCustomHotKey
CanAdd	<Table>_BeforeAddRow
CanDelete	<Table>_BeforeDeleteRow
Ctrl	IndexingFamily_PressCustomHotKey
CurrentRow	<Table>_BeforeAddRow <Table>_AfterAddRow <Table>_BeforeDeleteRow <Table>_AfterDeleteRow IndexingFamily_PressCustomHotKey
Handled	IndexingFamily_PressCustomHotKey
KeyCode	IndexingFamily_PressCustomHotKey
Shift	IndexingFamily_PressCustomHotKey

6.7.4 Matrix of Methods

Table 6-139: Matrix of Methods

Method	Parameters	Object
AddRow	*in RowNumber: Long*	DpDocArray
AssignDocField	*Source : DpDocField*	DpDocField
DeleteRow	*in RowNumber: Long* *out Result: Boolean*	DpDocArray
GetParamValue	*ParamName: string* *Return: String*	DpDocField
GetPreviousDocument	*Return: DpDocument*	DpDocument
GetNextDocument	*Return: DpDocument*	DpDocument

Method	Parameters	Object
GetSecondaryValue	*RowNumber: Long* *Direction: ESearchDirection* *Position: Long* *Return: DpDocField*	DpDocArrayField
GetPreviousFolder	*Return: DpFolder*	DpFolder
GetNextFolder	*Return: DpFolder*	DpFolder
GetSecondaryValueCount	*RowNumber: Long* *Direction: ESearchDirection* *Return: Long*	DpDocArrayField
GetTemplateByID	*ID: Integer* *Return: DpTemplate*	DpTemplates
IsParagraphHeader	*in RowNumber: Long* *return : Boolean*	DpDocArray
IsUserRow	*in RowNumber: Long* *return : Boolean*	DpDocArray
RemoveCharacter	*in Index: Long* *out Result: Boolean*	DpDocField
RequiresValidation	None	–
SetBounds	*in X: Long* *in Y: Long* *in W: Long* *in H: Long*	DpDocField
SetFocus	None	DpDocField
SetParamValue	*ParamName: string* *ParamValue: string*	DpDocField
SetStatusOK	None	DpDocField
SetStatusError	None	DpDocField
SkipRecognition	None	DpDocument DpDocField DpDocArrayField

6.7.5 Matrix of Object Model Elements

This section contains matrix tables on accessing Object Model elements from the different events in the Event Model.

6.7.5.1 Symbols

The following symbols are used in the matrix tables:

- CL = Classification
- RE = Recognition
- Empty box = N/A (no information available)
- R = Read-only
- RW = Read-Write
- W = Write only
- E = Empty
- N = Nothing
- For collections, Item() or Item(pos:integer) appears in the list of properties.

6.7.5.2 DpAdditionalInformation

This section contains the following DpAdditionalInformation matrix tables:

6.7.5.2.1 DpAdditionalInformation Project

Table 6-140: DpAdditionalInformation Project

Property	Item(Key)	
Type	String	
State	CL	RE
Project_Initialize		
Project_BeginTask		
Project_BeforeClassification		
Project_BeforeDocumentClassification		
Project_AfterDocumentClassification		
Project_AfterClassification		
Project_EndTask		
Project_Finalize		

Note: An empty box means that there is no information available for this property state.

6.7.5.2.2 DpAdditionalInformation Index Family

Table 6-141: DpAdditionalInformation Index Family

Property	Item(Key)	
Type	String	
State	CL	RE
IndexingFamily_Initialize		
IndexingFamily_BeforeDocumentRecognition		
<Field>_BeforeRecognition		
<Field>_AfterRecognition		R
IndexingFamily_AfterDocumentRecognition		
IndexingFamily_EnterDocument		
<Table>_BeforeAddRow		
<Table>_AfterAddRow		
<Table>_BeforeDeleteRow		
<Table>_AfterDeleteRow		
IndexingFamily_DocumentValidated		
IndexingFamily_ControlDocument		
IndexingFamily_PressCustomHotKey		
IndexingFamily_ExitDocument		
IndexingFamily_Finalize		

Note: An empty box means that there is no information available for this property state.

6.7.5.3 DpArray

This section contains the following DpArray matrix tables:

6.7.5.3.1 DpArray Project

Table 6-142: DpArray Project

Property	Count		Item()	
Type	Integer		`DpField`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification				
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification				
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.3.2 DpArray Index Family

Table 6-143: DpArray Index Family

Property	Count		Item()	
Type	Integer		`DpField`	
State	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R

Property	Count		Item()	
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.4 DpArrays

This section contains the following DpArrays matrix tables:

6.7.5.4.1 DpArrays Project

Table 6-144: DpArrays Project

Property	Count		Item()	
Type	Integer		String	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification				
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification				
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.4.2 DpArrays Index Family

Table 6-145: DpArrays Index Family

Property	Count		Item()	
Type	Integer		String	
State	CL	RE	CL	RE
IndexingFamily_Initialize		R	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R
<Field>_AfterRecognition	R	R	R	R

Property	Count		Item()	
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.5 DpBatch

This section contains the following DpBatch matrix tables:

6.7.5.5.1 DpBatch Project

Table 6-146: DpBatch Project

Property	Name		ID		Folders		ExternalValues	
Type	String		String		`DpFolders`		`DpExternalValues`	
State	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize								
Project_BeginTask								
Project_BeforeClassification	R		R		R		R	
Project_BeforeDocumentClassification	R		R		R		R	
Project_AfterDocumentClassification	R		R		R		R	
Project_AfterClassification	R		R		R		R	
Project_EndTask								
Project_Finalize								

Note: An empty box means that there is no information available for this property state.

6.7.5.5.2 DpBatch Index Family

Table 6-147: DpBatch Index Family

Property	Name		ID		Folders		ExternalValues	
Type	String		String		`DpFolders`		`DpExternalValues`	
State	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize								
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R	R	R	R	R
<Field>_AfterRecognition	R	R	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R	R	R
IndexingFamily_EnterDocument								
<Table>_BeforeAddRow								
<Table>_AfterAddRow								
<Table>_BeforeDeleteRow								

Property	Name		ID		Folders		ExternalValues	
<Table>_AfterDeleteRow								
IndexingFamily_DocumentValidated								
IndexingFamily_ControlDocument	R	R	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey								
IndexingFamily_ExitDocument								
IndexingFamily_Finalize								

Note: An empty box means that there is no information available for this property state.

6.7.5.6 DpCharacter

This section contains the following DpCharacter matrix tables:

6.7.5.6.1 DpCharacter Project

Table 6-148: DpCharacter Project

Property	CharacterHypotheses		BestValue (Deprecated)		BestConfidence (Deprecated)		SecondValue (Deprecated)		SecondConfidence (Deprecated)		Bounds	
Type	DpCharacterHypothesis		Byte		Long		Byte		Long		SRect	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize												
Project_BeginTask												
Project_BeforeClassification												
Project_BeforeDocumentClassification												
Project_AfterDocumentClassification												
Project_AfterClassification												

Property	CharacterHypotheses		BestValue (Deprecated)		BestConfidence (Deprecated)		SecondValue (Deprecated)		SecondConfidence (Deprecated)		Bounds	
Project_EndTask												
Project_Finalize												

Note: An empty box means that there is no information available for this property state.

6.7.5.6.2 DpCharacter Index Family

Table 6-149: DpCharacter Index Family

Property	CharacterHypotheses		BestValue (Deprecated)		BestConfidence (Deprecated)		SecondValue (Deprecated)		Bounds	
Type	DpCharacterHypoBytheses		Byte		Long		Byte		SRect	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize										
IndexingFamily_BeforeDocumentRecognition										
<Field>_BeforeRecognition										

Property	CharacterHypotheses		BestValue (Deprecated)		BestConfidence (Deprecated)		SecondValue (Deprecated)		Bounds	
<Field>_AfterRecognition	R*	R*	R*	R*	R*	R*	R*	R*	R*	R*
IndexingFamily_AfterDocumentRecognition										
IndexingFamily_EnterDocument										
<Table>_BeforeAddRow										
<Table>_AfterAddRow										
<Table>_BeforeDeleteRow										
<Table>_AfterDeleteRow										
IndexingFamily_DocumentValidated										

Property	CharacterHypotheses		BestValue (Deprecated)		BestConfidence (Deprecated)		SecondValue (Deprecated)		Bounds	
IndexingFamily_ControlDocument										
IndexingFamily_PressCustomHotKey										
IndexingFamily_ExitDocument										
IndexingFamily_Finalize										

* Only available for `CurrentField` or `CurrentArrayField` objects.

Note: An empty box means that there is no information available for this property state.

6.7.5.7 DpCharacters

This section contains the following DpCharacters matrix tables:

6.7.5.7.1 DpCharacters Project

Table 6-150: DpCharacters Project

Property	Count		Item()	
Type	Integer		`DpCharacter`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				

Property	Count		Item()	
Project_BeforeClassification				
Project_BeforeDocumentClassification				
Project_AfterDocumentClassification				
Project_AfterClassification				
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.7.2 DpCharacters Index Family

Table 6-151: DpCharacters Index Family

Property	Count		Item()	
Type	Integer		`DpCharacter`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition				
<Field>_BeforeRecognition				
<Field>_AfterRecognition	R*	R*	R*	R*
IndexingFamily_AfterDocumentRecognition				
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				

Property	Count		Item()	
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument				
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

* Only available for `CurrentField` or `CurrentArrayField` objects.

Note: An empty box means that there is no information available for this property state.

6.7.5.8 DpCharacterHypothesis

This section contains the following DpCharacterHypothesis matrix tables:

6.7.5.8.1 DpCharacterHypothesis Project

Table 6-152: DpCharacterHypothesis Project

Property	Value		Confidence	
Type	String		Integer	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification				
Project_BeforeDocumentClassification				

Property	Value		Confidence	
Project_AfterDocumentClassification				
Project_AfterClassification				
Project_EndTask				
Project_Finalize				

6.7.5.8.2 DpCharacterHypothesis Index Family

Table 6-153: DpCharacterHypothesis Index Family

Property	Value		Confidence	
Type	String		Integer	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition				
<Field>_BeforeRecognition				
<Field>_AfterRecognition	R*	R*	R*	R*
IndexingFamily_AfterDocumentRecognition				
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				

Property	Value		Confidence	
IndexingFamily_ControlDocument				
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

* Only available for `CurrentField` or `CurrentArrayField` objects.

Note: An empty box means that there is no information available for this property state.

6.7.5.9 DpCharacterHypotheses

This section contains the followingDpCharacterHypotheses matrix tables:

6.7.5.9.1 DpCharacterHypotheses Project

Table 6-154: DpCharacterHypotheses Project

Property	Count		Item()	
Type	Integer		`DpCharacterHypotheis`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification				
Project_BeforeDocumentClassification				
Project_AfterDocumentClassification				
Project_AfterClassification				
Project_EndTask				
Project_Finalize				

6.7.5.9.2 DpCharacterHypotheses Index Family

Table 6-155: DpCharacterHypotheses Index Family

Property	Count		Item()	
Type	Integer		DpCharacterHypothesis	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition				
<Field>_BeforeRecognition				
<Field>_AfterRecognition	R*	R*	R*	R*
IndexingFamily_AfterDocumentRecognition				
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument				
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

6.7.5.10 DpDocArray

This section contains the following DpDocArray matrix tables:

6.7.5.10.1 DpDocArray Project

Table 6-156: DpDocArray Project

Property	Count, RowCount		CurrentRow		Item()	
Type	Integer		Integer		`DpDocArrayField`	
State	CL	RE	CL	RE	CL	RE
Project_Initialize						
Project_BeginTask						
Project_BeforeClassification						
Project_BeforeDocumentClassification						
Project_AfterDocumentClassification						
Project_AfterClassification						
Project_EndTask						
Project_Finalize						

Note: An empty box means that there is no information available for this property state.

6.7.5.10.2 DpDocArray Index Family

Table 6-157: DpDocArray Index Family

Property	Count, RowCount		CurrentRow		Item()	
Type	Integer		Integer		`DpDocArrayField`	
State	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize						
IndexingFamily_BeforeDocumentRecognition		R		R		R
<Field>_BeforeRecognition		R		R		R
<Field>_AfterRecognition		R		R		R
IndexingFamily_AfterDocumentRecognition		R		R		R
IndexingFamily_EnterDocument						
<Table>_BeforeAddRow						
<Table>_AfterAddRow						
<Table>_BeforeDeleteRow						
<Table>_AfterDeleteRow						
IndexingFamily_DocumentValidated						

Property	Count, RowCount		CurrentRow		Item()	
IndexingFamily_ControlDocument		R		R		R
IndexingFamily_PressCustomHotKey						
IndexingFamily_ExitDocument						
IndexingFamily_Finalize						

Note: An empty box means that there is no information available for this property state.

6.7.5.11 DpDocArrays

This section contains the following DpDocArrays matrix tables:

6.7.5.11.1 DpDocArrays Project

Table 6-158: DpDocArrays Project

Property	Count		Item()	
Type	Integer		`DpDocArray`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	0		N	
Project_BeforeDocumentClassification	0		N	
Project_AfterDocumentClassification	0		N	
Project_AfterClassification	0		N	
Project_EndTask				

Property	Count		Item()	
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.11.2 DpDocArrays Index Family

Table 6-159: DpDocArrays Index Family

Property	Count		Item()	
Type	Integer		`DpDocArray`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition		R		R
<Field>_BeforeRecognition		R		R
<Field>_AfterRecognition		R		R
IndexingFamily_AfterDocumentRecognition		R		R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	0	R	N	R

Property	Count		Item()	
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.12 DpDocArrayField

This section contains the following DpDocArrayField matrix tables:

6.7.5.12.1 DpDocArrayField Project

Table 6-160: DpDocArrayField Project

Property	Count, Recognized, ReadOnly		Item(pos: integer)	
Type			`DpDocField`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification				
Project_BeforeDocumentClassification				
Project_AfterDocumentClassification				
Project_AfterClassification				
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.12.2 DpDocArrayField Index Family

Table 6-161: DpDocArrayField Index Family

Property	Count, Recognized, ReadOnly		Item(pos: integer)	
Type			`DpDocField`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition		R		R
<Field>_BeforeRecognition		R		R
<Field>_AfterRecognition		R		R
IndexingFamily_AfterDocumentRecognition		R		R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument		R		R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.13 DpDocument

This section contains the following DpDocument matrix tables:

6.7.5.13.1 DpDocument Project

Due to the large number of elements, the following matrix has been divided into multiple tables to accommodate all required columns.

Table 6-162: DpDocument Project

Property	ID		Pages		Template		TemplateProperties		TemplateHypotheses	
Type	String		`DpPages`		`DpTemplate`		`DpDocTemplate`		`DpTemplateHypotheses`	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize										
Project_BeginTask										
Project_BeforeClassification	R		R		R/W		R		E	
Project_BeforeDocumentClassification	R		R		R/W		R/W		R	
Project_AfterDocumentClassification	R		R		R/W		R/W		R	
Project_AfterClassification	R		R		R/W		R/W		R	

Property	ID		Pages		Template		TemplateProperties		TemplateHypotheses	
Project_EndTask										
Project_Finalize										

Note: An empty box means that there is no information available for this property state.

Table 6-163: DpDocument Project (continued)

Property	Folder		RankInFolder		Fields		RectifiedPaperOrientation		Arrays	
Type	`DpFolder`		Long		`DpDocFields`		EPaperOrientation		`DpDocArrays`	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize										
Project_BeginTask										
Project_BeforeClassification	R		R		R		R		E	
Project_BeforeDocumentClassification	R		R		R		R		E	
Project_AfterDocumentClassification	R		R		R		R		E	
Project_AfterClassification	R		R		R		R		E	

Property	Folder		RankInFolder		Fields		RectifiedPaperOrientation		Arrays	
Project_EndTask										
Project_Finalize										

Note: An empty box means that there is no information available for this property state.

Table 6-164: DpDocument Project (continued)

Property	Recognized		Classified		Rejected		RectifiedDuplexinversion		Status	
Type	Boolean		Boolean		Boolean		Boolean		EControlStatus	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize										
Project_BeginTask										
Project_BeforeClassification	R		R		R		R		R	
Project_BeforeDocumentClassification	R		R		R		R		R	
Project_AfterDocumentClassification	R		R		R		R		R	
Project_AfterClassification	R		R		R		R		R	

Property	Recognized		Classified		Rejected		RectifiedDuplexinversion		Status	
Project_EndTask										
Project_Finalize										

Note: An empty box means that there is no information available for this property state.

Table 6-165: DpDocument Project (continued)

Property	IndexingFamily		ExternalValues	
Type	`DpIndexingFamily`		`DpExternalValues`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.13.2 DpDocument Index Family

Due to the large number of elements, the following matrix has been divided into multiple tables to accommodate all required columns.

Table 6-166: DpDocument Index Family

Property	ID		Pages		Template		TemplateProperties		TemplateHypotheses	
Type	String		`DpPages`		`DpTemplate`		`DpDocTemplate`		`DpTemplateHypotheses`	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize										
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	N	R	R	R	E	R
<Field>_BeforeRecognition	R	R	R	R	N	R	R	R	E	R
<Field>_AfterRecognition	R	R	R	R	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R	R	R	R	R
IndexingFamily_EnterDocument										

Property	ID		Pages		Template		TemplateProperties		TemplateHypotheses	
<Table>_BeforeAddRow										
<Table>_AfterAddRow										
<Table>_BeforeDeleteRow										
<Table>_AfterDeleteRow										
IndexingFamily_DocumentValidated										
IndexingFamily_ControlDocument	R	R	R	R	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey										
IndexingFamily_ExitDocument										

Property	ID		Pages		Template		TemplateProperties		TemplateHypotheses	
IndexingFamily_Finalize										

Note: An empty box means that there is no information available for this property state.

Table 6-167: DpDocument Index Family (continued)

Property	Folder		RankInFolder		Fields		RectifiedPaperOrientation		Arrays	
Type	`DpFolder`		Long		`DpDocFields`		EPaperOrientation		`DpDocArrays`	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize										
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R	R	R	E	R
<Field>_BeforeRecognition	R	R	R	R	R	R	R	R	E	R
<Field>_AfterRecognition	R	R	R	R	R	R	R	R	E	R

Property	Folder		RankInFolder		Fields		RectifiedPaperOrientation		Arrays	
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R	R	R	E	R
IndexingFamily_EnterDocument										
<Table>_BeforeAddRow										
<Table>_AfterAddRow										
<Table>_BeforeDeleteRow										
<Table>_AfterDeleteRow										
IndexingFamily_DocumentValidated										
IndexingFamily_ControlDocument	R	R	R	R	R	R	R	R	E	R

Property	Folder		RankInFolder		Fields		RectifiedPaperOrientation		Arrays	
IndexingFamily_PressCustomHotKey										
IndexingFamily_ExitDocument										
IndexingFamily_Finalize										

Note: An empty box means that there is no information available for this property state.

Table 6-168: DpDocument Index Family (continued)

Property	Recognized		Classified		Rejected		RectifiedDuplexinversion		Status	
Type	Boolean		Boolean		Boolean		Boolean		EControlStatus	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize										
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R	R	R	R	R

Property	Recognized		Classified		Rejected		RectifiedDuplexinversion		Status	
<Field>_BeforeRecognition	R	R	R	R	R	R	R	R	R	R
<Field>_AfterRecognition	R	R	R	R	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R	R	R	R	R
IndexingFamily_EnterDocument										
<Table>_BeforeAddRow										
<Table>_AfterAddRow										
<Table>_BeforeDeleteRow										
<Table>_AfterDeleteRow										

Property	Recognized		Classified		Rejected		RectifiedDuplexinversion		Status	
IndexingFamily_DocumentValidated										
IndexingFamily_ControlDocument	R	R	R	R	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey										
IndexingFamily_ExitDocument										
IndexingFamily_Finalize										

Note: An empty box means that there is no information available for this property state.

Table 6-169: DpDocument Index Family (continued)

Property	IndexingFamily		ExternalValues	
Type	`DpIndexingFamily`		`DpExternalValues`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	N	R	R	R

Property	IndexingFamily		ExternalValues	
<Field>_BeforeRecognition	N	R	R	R
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.14 DpDocuments

This section contains the following DpDocuments matrix tables:

6.7.5.14.1 DpDocuments Project

Table 6-170: DpDocuments Project

Property	Count		Item()	
Type	Integer		`DpDocument`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.14.2 DpDocuments Index Family

Table 6-171: DpDocuments Index Family

Property	Count		Item()	
Type	Integer		`DpDocument`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R

Property	Count		Item()	
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.15 DpDocField

This section contains the following DpDocField matrix tables:

6.7.5.15.1 DpDocField Project

Due to the large number of elements, the following matrix has been divided into multiple tables to accommodate all required columns.

Table 6-172: DpDocField Project

Property	Field		Value		Confidence		Bounds		Status	
Type	`DpField`		String		Integer		sRec		EControlStatus	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize										
Project_BeginTask										
Project_BeforeClassification	R		R/W		R		R		R	
Project_BeforeDocumentClassification	R		R/W		R		R		R	
Project_AfterDocumentClassification	R		R/W		R		R		R	
Project_AfterClassification	R		R/W		R		R		R	
Project_EndTask										
Project_Finalize										

Note: An empty box means that there is no information available for this property state.

Table 6-173: DpDocField Project (continued)

Property	Recognized		RequiresValidation		ReadOnly	
Type	Boolean		Boolean		Boolean	
State	CL	RE	CL	RE	CL	RE
Project_Initialize						
Project_BeginTask						
Project_BeforeDocumentClassification	R		R/W		R/W	
Project_AfterDocumentClassification	R		R/W		R/W	
Project_AfterDocumentClassification	R		R/W		R/W	
Project_AfterClassification	R		R/W		R/W	
Project_EndTask						
Project_Finalize						

Note: An empty box means that there is no information available for this property state.

Table 6-174: DpDocField Project (continued)

Property	OutputMessage		OCREngineOutput	
Type	String		`DpCharacters`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R/W		E	

Property	OutputMessage		OCREngineOutput	
Project_BeforeDocumentClassification	R/W		E	
Project_AfterDocumentClassification	R/W		R	
Project_AfterClassification	R/W		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.15.2 DpDocField Index Family

Due to the large number of elements, the following matrix has been divided into multiple tables to accommodate all required columns.

Table 6-175: DpDocField Index Family

Property	Field		Value		Confidence		Bounds		Status	
Type	DpField		String		Integer		sRec		EControlStatus	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize										
IndexingFamily_BeforeDocumentRecognition	R	R	R/W	R/W	R	R	R	R	R	R
<Field>_BeforeRecognition	R	R	R/W	R/W	R	R	R	R	R	R

Property	Field		Value		Confidence		Bounds		Status	
<Field>_AfterRecognition	R	R	R/W	R/W	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R/W	R/W	R	R	R	R	R	R
IndexingFamily_EnterDocument										
<Table>_BeforeAddRow										
<Table>_AfterAddRow										
<Table>_BeforeDeleteRow										
<Table>_AfterDeleteRow										
IndexingFamily_DocumentValidated										

Property	Field		Value		Confidence		Bounds		Status	
IndexingFamily_ControlDocument	R	R	R/W	R/W	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey										
IndexingFamily_ExitDocument										
IndexingFamily_Finalize										

Note: An empty box means that there is no information available for this property state.

Table 6-176: DpDocField Index Family (continued)

Property	Recognized		RequiresValidation		ReadOnly	
Type	Boolean		Boolean		Boolean	
State	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize						
IndexingFamily_BeforeDocumentRecognition	R	R	R/W	R/W	R/W	R/W
<Field>_BeforeRecognition	R	R	R/W	R/W	R/W	R/W

Property	Recognized		RequiresValidation		ReadOnly	
<Field>_AfterRecognition	R	R	R/W	R/W	R/W	R/W
IndexingFamily_AfterDocumentRecognition	R	R	R/W	R/W	R/W	R/W
IndexingFamily_EnterDocument						
<Table>_BeforeAddRow						
<Table>_AfterAddRow						
<Table>_BeforeDeleteRow						
<Table>_AfterDeleteRow						
IndexingFamily_DocumentValidated						
IndexingFamily_ControlDocument	R	R	R/W	R/W	R/W	R/W
IndexingFamily_PressCustomHotKey						
IndexingFamily_ExitDocument						
IndexingFamily_Finalize						

Note: An empty box means that there is no information available for this property state.

Table 6-177: DpDocField Index Family (continued)

Property	OutputMessage		OCREngineOutput	
Type	String		`DpCharacters`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	R/W	R/W	R	R
<Field>_BeforeRecognition	R/W	R/W	R	R
<Field>_AfterRecognition	R/W	R/W	R	R
IndexingFamily_AfterDocumentRecognition	R/W	R/W	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R/W	R/W	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.16 DpDocFields

This section contains the following DpDocFields matrix tables:

6.7.5.16.1 DpDocFields Project

Table 6-178: DpDocFields Project

Property	Count		Item()	
Type	Integer		`DpDocField`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.16.2 DpDocFields Index Family

Table 6-179: DpDocFields Index Family

Property	Count		Item()	
Type	Integer		`DpDocField`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R

Property	Count		Item()	
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.17 DpDocTemplate

This section contains the following DpDocTemplate matrix tables:

6.7.5.17.1 DpDocTemplate Project

Table 6-180: DpDocTemplate Project

Property	Template		Confidence		PaperOrientation		PreClassificationRate		DuplexInversion	
Type	`DpTemplate`		Long		EPaperOrientation		Long		Boolean	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize										
Project_BeginTask										
Project_BeforeClassification	R		R		R		R		R	
Project_BeforeDocumentClassification	R		R		R		R		R	
Project_AfterDocumentClassification	R		R		R		R		R	
Project_AfterClassification	R		R		R		R		R	
Project_EndTask										
Project_Finalize										

Note: An empty box means that there is no information available for this property state.

6.7.5.17.2 DpDocTemplate Index Family

Table 6-181: DpDocTemplate Index Family

Property	Template		Confidence		PaperOrientation		PreClassificationRate		DuplexInversion	
Type	`DpTemplate`		Long		EPaperOrientation		Long		Boolean	
State	CL	RE	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize										
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R	R	R	R	R	R	R
<Field>_AfterRecognition	R	R	R	R	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R	R	R	R	R
IndexingFamily_EnterDocument										

Property	Template		Confidence		PaperOrientation		PreClassificationRate		DuplexInversion	
<Table>_BeforeAddRow										
<Table>_AfterAddRow										
<Table>_BeforeDeleteRow										
<Table>_AfterDeleteRow										
IndexingFamily_DocumentValidated										
IndexingFamily_ControlDocument	R	R	R	R	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey										
IndexingFamily_ExitDocument										

Property	Template		Confidence		PaperOrientation		PreClassificationRate		DuplexInversion	
IndexingFamily_Finalize										

Note: An empty box means that there is no information available for this property state.

6.7.5.18 DpTemplateHypotheses

This section contains the following DpTemplateHypotheses matrix tables:

6.7.5.18.1 DpTemplateHypotheses Project

Table 6-182: DpTemplateHypotheses Project

Property	Count		Item()	
Type	Integer		`DpDocTemplate`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.18.2 DpTemplateHypotheses Index Family

Table 6-183: DpTemplateHypotheses Index Family

Property	Count		Item()	
Type	Integer		`DpDocTemplate`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.19 DpExternalValue

This section contains the following DpExternalValue matrix tables:

6.7.5.19.1 DpExternalValue Project

Table 6-184: DpExternalValue Project

Property	Name		Value	
Type	String		String	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R/W	
Project_BeforeDocumentClassification	R		R/W	
Project_AfterDocumentClassification	R		R/W	
Project_AfterClassification	R		R/W	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.19.2 DpExternalValue Index Family

Table 6-185: DpExternalValue Index Family

Property	Name		Value	
Type	String		String	
State	CL	RE	CL	RE
IndexingFamily_Initialize				

Property	Name		Value	
IndexingFamily_BeforeDocumentRecognition	R	R	R/W	R/W
<Field>_BeforeRecognition	R	R	R/W	R/W
<Field>_AfterRecognition	R	R	R/W	R/W
IndexingFamily_AfterDocumentRecognition	R	R	R/W	R/W
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R/W	R/W
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.20 DpExternalValues

This section contains the following DpExternalValues matrix tables:

6.7.5.20.1 DpExternalValues Project

Table 6-186: DpExternalValues Project

Property	Count		Item()	
Type	Integer		`DpExternalValue`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.20.2 DpExternalValues Index Family

Table 6-187: DpExternalValues Index Family

Property	Count		Item()	
Type	Integer		`DpExternalValue`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R

Property	Count		Item()	
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.21 DpField

This section contains the following DpField matrix tables:

6.7.5.21.1 DpField Project

Table 6-188: DpField Project

Properties	ID		Name		Caption		Parameters	
Type	String		String		String		DpParameters	
States	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize								
Project_BeginTask								
Project_BeforeClassification								
Project_BeforeDocumentClassification	R		R		R/W		R	
Project_AfterDocumentClassification	R		R		R/W		R	
Project_AfterClassification								
Project_EndTask								
Project_Finalize								

Note: An empty box means that there is no information available for this property state.

6.7.5.21.2 DpField Index Family

Table 6-189: DpField Index Family

Property	ID		Name		Caption		Parameters	
Type	String		String		String		`DpParameters`	
State	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R	R/W	R/W	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R	R	R	R	R
<Field>_AfterRecognition	R	R	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R	R	R
IndexingFamily_EnterDocument								
<Table>_BeforeAddRow								
<Table>_AfterAddRow								
<Table>_BeforeDeleteRow								

Property	ID		Name		Caption		Parameters	
<Table>_AfterDeleteRow								
IndexingFamily_DocumentValidated								
IndexingFamily_ControlDocument	R	R	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey								
IndexingFamily_ExitDocument								
IndexingFamily_Finalize	R	R	R	R	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.22 DpFields

This section contains the following DpFields matrix tables:

6.7.5.22.1 DpFields Project

Table 6-190: DpFields Project

Property	Count		Item()	
Type	Integer		`DpField`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification				
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification				
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.22.2 DpFields Index Family

Table 6-191: DpFields Index Family

Property	Count		Item()	
Type	Integer		`DpField`	
State	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R
<Field>_AfterRecognition	R	R	R	R

Property	Count		Item()	
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.23 DpFolder

This section contains the following DpFolder matrix tables:

6.7.5.23.1 DpFolder Project

Table 6-192: DpFolder Project

Property	Documents		ExternalValues	
Type	`DpDocuments`		`DpExternalValues`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.23.2 DpFolder Index Family

Table 6-193: DpFolder Index Family

Property	Documents		ExternalVaues	
Type	`DpDocuments`		`DpExernalValues`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R
<Field>_AfterRecognition	R	R	R	R

Property	Documents		ExternalVaues	
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument		R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.24 DpFolders

This section contains the following DpFolders matrix tables:

6.7.5.24.1 DpFolders Project

Table 6-194: DpFolders Project

Property	Count		Item()	
Type	Integer		`DpFolder`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.24.2 DpFolders Index Family

Table 6-195: DpFolders Index Family

Property	Count		Item()	
Type	Integer		`DpFolder`	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R
<Field>_AfterRecognition	R	R	R	R

Property	Count		Item()	
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.25 DpIndexingFamily

This section contains the following DpIndexingFamily matrix tables:

6.7.5.25.1 DpIndexingFamily Project

Table 6-196: DpIndexingFamily Project

Property	Name		Fields		Arrays	
Type	String		`DpFields`		`DpArrays`	
State	CL	RE	CL	RE	CL	RE
Project_Initialize						
Project_BeginTask						
Project_BeforeClassification						
Project_BeforeDocumentClassification	R		R		R	
Project_AfterDocumentClassification	R		R		R	
Project_AfterClassification						
Project_EndTask						
Project_Finalize						

Note: An empty box means that there is no information available for this property state.

6.7.5.25.2 DpIndexingFamily Index Family

Table 6-197: DpIndexingFamily Index Family

Property	Name		Fields		Arrays	
Type	String		`DpFields`		`DpArrays`	
State	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R	R	R

Property	Name		Fields		Arrays	
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R	R	R
<Field>_AfterRecognition	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R
IndexingFamily_EnterDocument						
<Table>_BeforeAddRow						
<Table>_AfterAddRow						
<Table>_BeforeDeleteRow						
<Table>_AfterDeleteRow						
IndexingFamily_DocumentValidated						
IndexingFamily_ControlDocument	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey						

Property	Name		Fields		Arrays	
IndexingFamily_ExitDocument						
IndexingFamily_Finalize	R	R	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.26 DpIndexingFamilies

This section contains the following DpIndexingFamilies matrix tables:

6.7.5.26.1 DpIndexingFamilies Project

Table 6-198: DpIndexingFamilies Project

Property	Count		Item()	
Type	Integer		`DpIndexingFamily`	
State	CL	RE	CL	RE
Project_Initialize	0	0	N	N
Project_BeginTask	0	0	N	N
Project_BeforeClassification	0		N	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	0		N	
Project_EndTask	0	0	N	N
Project_Finalize	0	0	N	N

Note: An empty box means that there is no information available for this property state.

6.7.5.26.2 DpIndexingFamilies Index Family

Table 6-199: DpIndexingFamilies Index Family

Property	Count		Item()	
Type	Integer		`DpIndexingFamily`	
State	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.27 DpLineItems

There are no matrix tables for this element because `DpLineItems` objects are accessed only in VBA relation scripts that are used for the Line Item Free Form Engine (*LIFFE*).

6.7.5.28 DpPage

This section contains the following DpPage matrix tables:

6.7.5.28.1 DpPage Project

Table 6-200: DpPage Project

Property	FilePath	
Type	String	
State	CL	RE
Project_Initialize		
Project_BeginTask		
Project_BeforeClassification	R	
Project_BeforeDocumentClassification	R	
Project_AfterDocumentClassification	R	
Project_AfterClassification	R	
Project_EndTask		
Project_Finalize		

Note: An empty box means that there is no information available for this property state.

6.7.5.28.2 DpPage Index Family

Table 6-201: DpPage Index Family

Property	FilePath	
Type	String	
State	CL	RE
IndexingFamily_Initialize		
IndexingFamily_BeforeDocumentRecognition	R	R
<Field>_BeforeRecognition	R	R
<Field>_AfterRecognition	R	R
IndexingFamily_AfterDocumentRecognition	R	R
IndexingFamily_EnterDocument		
<Table>_BeforeAddRow		
<Table>_AfterAddRow		
<Table>_BeforeDeleteRow		
<Table>_AfterDeleteRow		
IndexingFamily_DocumentValidated		
IndexingFamily_ControlDocument	R	R
IndexingFamily_PressCustomHotKey		
IndexingFamily_ExitDocument		
IndexingFamily_Finalize		

Note: An empty box means that there is no information available for this property state.

6.7.5.29 DpPages

This section contains the following DpPages matrix tables:

6.7.5.29.1 DpPages Project

Table 6-202: DpPages Project

Property	Count		Item()	
Type	Integer		DpPage	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R	
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.29.2 DpPages Index Family

Table 6-203: DpPages Index Family

Property	Count		Item()	
Type	Integer		DpPage	
State	CL	RE	CL	RE
IndexingFamily_Initialize				
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R

Property	Count		Item()	
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.30 DpParameter

This section contains the following DpParameter matrix tables:

6.7.5.30.1 DpParameter Project

Table 6-204: DpParameter Project

Property	Name	
Type	String	
State	CL	RE
Project_Initialize		
Project_BeginTask		
Project_BeforeClassification		
Project_BeforeDocumentClassification	R	
Project_AfterDocumentClassification	R	
Project_AfterClassification		
Project_EndTask		
Project_Finalize		

Note: An empty box means that there is no information available for this property state.

6.7.5.30.2 DpParameter Index Family

Table 6-205: DpParameter Index Family

Property	Name	
Type	String	
State	CL	RE
IndexingFamily_Initialize	R	R
IndexingFamily_BeforeDocumentRecognition	R	R
<Field>_BeforeRecognition	R	R
<Field>_AfterRecognition	R	R
IndexingFamily_AfterDocumentRecognition	R	R
IndexingFamily_EnterDocument		
<Table>_BeforeAddRow		
<Table>_AfterAddRow		
<Table>_BeforeDeleteRow		

Property	Name	
<Table>_AfterDeleteRow		
IndexingFamily_DocumentValidated		
IndexingFamily_ControlDocument	R	R
IndexingFamily_PressCustomHotKey		
IndexingFamily_ExitDocument		
IndexingFamily_Finalize	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.31 DpParameters

This section contains the following DpParameters matrix tables:

6.7.5.31.1 DpParameters Project

Table 6-206: DpParameters Project

Property	Count		Item()	
Type	Integer		`DpField`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification				
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification				
Project_EndTask				
Project_Finalize				

Note: An empty box means that there is no information available for this property state.

6.7.5.31.2 DpParameters Index Family

Table 6-207: DpParameters Index Family

Property	Count		Item()	
Type	Integer		`DpField`	
State	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				

Property	Count		Item()	
IndexingFamily_Finalize	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.32 DpProject

This section contains the following DpProject matrix tables:

6.7.5.32.1 DpProject Project

Table 6-208: DpProject Project

Property	Name		Version		IndexingFamilies		Templates	
Type	String		String		`DpIndexFamilies`		`DpTemplates`	
State	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize	R	R	R	R	E	E	E	E
Project_BeginTask	R	R	R	R	E	E	E	E
Project_BeforeClassification	R		R		E		R	
Project_BeforeDocumentClassification	R		R		R		R	
Project_AfterDocumentClassification	R		R		R		R	
Project_AfterClassification	R		R		E		R	
Project_EndTask	R	R	R	R	E	E	E	E

Property	Name		Version		IndexingFamilies		Templates	
Project_Finalize	R	R	R	R	E	E	E	E

Note: An empty box means that there is no information available for this property state.

6.7.5.32.2 DpProject Index Family

Table 6-209: DpProject Index Family

Property	Name		Version		IndexingFamilies		Templates	
Type	`DpProject`		`DpBatch`		`DpFolder`		`DpDocument`	
State	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R	R	R	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R	R	R	R	R
<Field>_AfterRecognition	R	R	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R	R	R
IndexingFamily_EnterDocument								
<Table>_BeforeAddRow								

Property	Name		Version		IndexingFamilies		Templates	
<Table>_AfterAddRow								
<Table>_BeforeDeleteRow								
<Table>_AfterDeleteRow								
IndexingFamily_DocumentValidated								
IndexingFamily_ControlDocument	R	R	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey								
IndexingFamily_ExitDocument								
IndexingFamily_Finalize	R	R	R	R	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.33 DpTask

This section contains the following DpTask matrix tables:

6.7.5.33.1 DpTask Project

Table 6-210: DpTask Project

Property	CurrentProject		CurrentBatch		CurrentFolder		CurrentDocument	
Type	`DpProject`		`DpBatch`		`DpFolder`		`DpDocument`	
State	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize	R	R	N	N	N	N	N	N
Project_BeginTask	R	R	N	N	N	N	N	N
Project_BeforeClassification	R		R		R		N	
Project_BeforeDocumentClassification	R		R		R		R	
Project_AfterDocumentClassification	R		R		R		R	
Project_AfterClassification	R		R		R		N	
Project_EndTask	R	R	N	N	N	N	N	N
Project_Finalize	R	R	N	N	N	N	N	N

Note: An empty box means that there is no information available for this property state.

Table 6-211: DpTask Project (continued)

Property	CurrentField		CurrentTask		CurrentArrayField		TaskType	
Type	`DpDocField`		`DpTask`		`DpDocArrayField`		ETaskType	
State	CL	RE	CL	RE	CL	RE	CL	RE
Project_Initialize	N	N	R	R	N	N	*0	*0
Project_BeginTask	N	N	R	R	N	N	*1	*3
Project_BeforeClassification	N		R		N		*1	
Project_BeforeDocumentClassification	N		R		N		*1	
Project_AfterDocumentClassification	N		R		N		*1	
Project_AfterClassification	N		R		N		*1	
Project_EndTask	N	N	R	R	N	N	*1	*3
Project_Finalize	N	N	R	R	N	N	*0	*0

Note: An empty box means that there is no information available for this property state.

*0 = R, ttUnknown

*1 = R, ttClassification

*3 = R, ttRecognition

6.7.5.33.2 DpTask Index Family

Table 6-212: DpTask Index Family

Property	CurrentProject		CurrentBatch		CurrentFolder		CurrentDocument	
Type	`DpProject`		`DpBatch`		`DpFolder`		`DpDocument`	
State	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	N	N	N	N	N	N
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R	R	R	R	R
<Field>_AfterRecognition	R	R	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R	R	R
IndexingFamily_EnterDocument								
<Table>_BeforeAddRow								
<Table>_AfterAddRow								
<Table>_BeforeDeleteRow								

Property	CurrentProject		CurrentBatch		CurrentFolder		CurrentDocument	
<Table>_AfterDeleteRow								
IndexingFamily_DocumentValidated								
IndexingFamily_ControlDocument	R	R	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey								
IndexingFamily_ExitDocument								
IndexingFamily_Finalize	R	R	N	N	N	N	N	N

Note: An empty box means that there is no information available for this property state.

Table 6-213: DpTask Index Family (continued)

Property	CurrentField		CurrentTask		CurrentArrayField		TaskType	
Type	`DpDocField`		`DpTask`		`DpDocArrayField`		*`ETaskType`*	
State	CL	RE	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize	N	N	R	R	N	N	*1	*3

Property	CurrentField		CurrentTask		CurrentArrayField		TaskType	
IndexingFamily_BeforeDocumentRecognition	N	N	R	R	N	N	*1	*3
<Field>_BeforeRecognition	R	R	R	R	R	R	*1	*3
<Field>_AfterRecognition	R	R	R	R	R	R	*1	*3
IndexingFamily_AfterDocumentRecognition	N	N	R	R	N	N	*1	*3
IndexingFamily_EnterDocument								
<Table>_BeforeAddRow								
<Table>_AfterAddRow								
<Table>_BeforeDeleteRow								
<Table>_AfterDeleteRow								
IndexingFamily_DocumentValidated								

Property	CurrentField		CurrentTask		CurrentArrayField		TaskType	
IndexingFamily_ControlDocument	N	N	R	R	N	N	*1	*3
IndexingFamily_PressCustomHotKey								
IndexingFamily_ExitDocument								
IndexingFamily_Finalize	N	N	R	R	N	N	*1	*3

Note: An empty box means that there is no information available for this property state.

*0 = R, ttUnknown

*1 = R, ttClassification

*3 = R, ttRecognition

6.7.5.34 DpTemplate

This section contains the following DpTemplate matrix tables:

6.7.5.34.1 DpTemplate Project

Table 6-214: DpTemplate Project

Property	Name, ID, Code, Separator		IndexingFamily	
Type	String		`DpIndexingFamily`	
State	CL	RE	CL	RE
Project_Initialize				
Project_BeginTask				
Project_BeforeClassification	R		R*	

Property	Name, ID, Code, Separator		IndexingFamily	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	R		R*	
Project_EndTask				
Project_Finalize				

* = Access to the index family is automatic, or the default value is Nothing if the template has no index family.

Note: An empty box means that there is no information available for this property state.

6.7.5.34.2 DpTemplate Index Family

Table 6-215: DpTemplate Index Family

Property	Name		Fields		Arrays	
Type	String		`DpFields`		`DpArrays`	
State	CL	RE	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R	R	R
<Field>_BeforeRecognition	R	R	R	R	R	R
<Field>_AfterRecognition	R	R	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R	R	R

Property	Name		Fields		Arrays	
IndexingFamily_EnterDocument						
<Table>_BeforeAddRow						
<Table>_AfterAddRow						
<Table>_BeforeDeleteRow						
<Table>_AfterDeleteRow						
IndexingFamily_DocumentValidated						
IndexingFamily_ControlDocument	R	R	R	R	R	R
IndexingFamily_PressCustomHotKey						
IndexingFamily_ExitDocument						
IndexingFamily_Finalize	R	R	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.5.35 DpTemplates

This section contains the following DpTemplates matrix tables:

6.7.5.35.1 DpTemplates Project

Table 6-216: DpTemplates Project

Property	Count		Item(), GetTemplateById()	
Type	Integer		`DpTemplate`	
State	CL	RE	CL	RE
Project_Initialize	0	0	N	N
Project_BeginTask	0	0	N	N
Project_BeforeClassification	0*		N*	
Project_BeforeDocumentClassification	R		R	
Project_AfterDocumentClassification	R		R	
Project_AfterClassification	0*		N*	
Project_EndTask	0	0	N	N
Project_Finalize	0	0	N	N

* = Access to a template is automatic. The default value is Nothing.

Note: An empty box means that there is no information available for this property state.

6.7.5.35.2 DpTemplates Index Family

Table 6-217: DpTemplates Index Family

Property	Count		Item(), GetTemplateById()	
Type	Integer		`DpIndexingFamily`	
State	CL	RE	CL	RE
IndexingFamily_Initialize	R	R	R	R
IndexingFamily_BeforeDocumentRecognition	R	R	R	R

Property	Count		Item(), GetTemplateById()	
<Field>_BeforeRecognition	R	R	R	R
<Field>_AfterRecognition	R	R	R	R
IndexingFamily_AfterDocumentRecognition	R	R	R	R
IndexingFamily_EnterDocument				
<Table>_BeforeAddRow				
<Table>_AfterAddRow				
<Table>_BeforeDeleteRow				
<Table>_AfterDeleteRow				
IndexingFamily_DocumentValidated				
IndexingFamily_ControlDocument	R	R	R	R
IndexingFamily_PressCustomHotKey				
IndexingFamily_ExitDocument				
IndexingFamily_Finalize	R	R	R	R

Note: An empty box means that there is no information available for this property state.

6.7.6 Matrix of Object Properties

Table 6-218: Matrix of Object Properties

Property	Type	Default value	Used in
AdditionalInformation	DpAdditionalInfomation	Empty	DpDocField
Arrays	DpDocArrays	Empty	DpDocument DpIndexingFamily
BestValue	Byte	Empty	DpCharacter: This property is not Unicode-compliant and has been deprecated. It remains in the DOM for backward-compatibility only. It is highly recommended that this property be replaced by the new property "Value" on page 336 (CharacterHypotheses).
BestConfidence	Long	0	DpCharacter: This property is not Unicode-compliant and has been deprecated. It remains in the DOM for backward-compatibility only. It is highly recommended that this property be replaced by the new property "DpCharacterHypothesis" on page 292.
Bounds	SRect	Empty	DpCharacter
Caption	String	""	DpField
CharacterHypotheses	Integer	0	DpCharacter
Classified	Boolean	False	DpDocument
Code	String	""	DpTemplate

Property	Type	Default value	Used in
Confidence DpCharacterHypothesis	Integer	0	DpCharacterHypothesis: This property replaces BestConfidence and SecondConfidence to provide Unicode support.
Confidence (DpDocField)	Integer	-1	DpDocField
Confidence (DpDocTemplate)	Long	0	DpDocTemplate
CurrentArrayfield	DpDocArrayField	Nothing	DpTask
CurrentBatch	DpBatch	Nothing	DpTask
CurrentDocument	DpDocument	Nothing	DpTask
CurrentField	DpDocField	Nothing	DpTask
CurrentFolder	DpFolder	Nothing	DpTask
CurrentProject	DpProject	Nothing	DpTask
CurrentRow	Integer	-1	DpDocArray
CurrentTask	DpTask	Self	DpTask
Documents	DpDocuments	Empty	DpFolder
DuplexInversion	Boolean	False	DpDocTemplate
ID	String	“”	DpBatch- Batch ID (used as a parameter in some *DFL* functions). DpDocument: Document ID. Used as a parameter in some *DFL* functions. DpField: Field ID as defined in the Project. DpTemplate: Template ID as defined in the Project.
ExternalValues	DpExternalValues	Empty	DpBatch DpDocument DpFolder
Field	DpField	Nothing	DpDocField

Property	Type	Default value	Used in
Fields	DpDocFields (`DpDocument`) DpFields(`DpIndexingFamily`)	Empty	DpDocument DpIndexingFamily
Filepath	String	""	DpPage
Folder	DpFolder	Nothing	DpDocument
Folder	DpFolders	Empty	DpBatch
IndexingFamily	`DpIndexFamily` (`DpDocument`) DpIndexingFamily (`DpTemplate`)	Nothing	DpDocument DpTemplate
IndexingFamilies	DpIndexingFamilies	Empty	DpProject
Name	String	""	DpArray DpBatch DpExternalValue DpField DpIndexingFamily DpParameter DpProject DpTemplate
OCREngineOutput	DpCharacters	Empty	DpDocField
OutputMessage	String	""	DpDocField
Pages	DpPages	Empty	DpDocument
PaperOrientation	EPaperOrientation	poNormal	DpDocTemplate
Parameters	DpParameter	Empty	DpField
PreClassificationRate	Long	0	DpDocTemplate
RankInFolder	Long	0	DpDocument
ReadOnly	Boolean	False	DpDocField
Recognized	Boolean	True for `DpDocField` False for `DpDocument`	DpDocField DpDocument
RectifiedDuplexInversion	Boolean	False	DpDocument

Property	Type	Default value	Used in
RectifiedPaperOrientation	EPaperOrientation	poNormal	DpDocument
Rejected	Boolean	False	DpDocument
RequiresValidation	Boolean	False	DpDocField
RowCount	Long	0	DpDocArray
SecondConfidence	Long	0 if no second value	DpCharacter: This property is not Unicode-compliant and has been deprecated. It remains in the *DOM* for backward-compatibility only. It is highly recommended that this property be replaced by the new property "Confidence (DpCharacterHypothesis)" on page 311.
SecondValue		Chr(0) if no second value	DpCharacter: This property is not Unicode-compliant and has been deprecated. It remains in the *DOM* for backward-compatibility only. It is highly recommended that this property be replaced by the new property "Value" on page 336DpCharacterHypothesis.
Separator	Byte	False	DpTemplate
Status	EControlStatus	csNone	DpDocField DpDocument
TaskType	ETaskType	ttUnknown	DpTask
Template	DpTemplate	Nothing	DpDocument DpDocTemplate
TemplateHypotheses	DpTemplateHypotheses	Empty	DpDocument

Property	Type	Default value	Used in
TemplateProperties	DpDocTemplate	Nothing	DpDocument
Templates	DpTemplates	Empty	DpProject
Value	String	""	DpDocField DpCharacterHypothesis DpExternalValue
Version	String	""	DpProject

Chapter 7

Client Side Scripting

7.1 Overview

This guide explains how you can create client scripts to automate tasks in Intelligent Capture processes.

7.1.1 Understanding Client-Side Scripting

A client-side script is a program that runs as part of an Intelligent Capture process. You can create script actions and associate them with specific events that are defined in each module. When the event occurs, your script action is executed.

For example, suppose you wanted to limit the number of pages within each document. You might create an action—or event handler—that displays a warning if more than five pages exist within a single node. Assigning this handler to the ScanPlus module event named `AfterPageAdded` would cause the warning to appear when a ScanPlus user adds the 6th node.

Scripts are written as Microsoft .NET Framework assemblies, typically using either Visual Basic or Visual C# as the programming language. You can create your assembly externally and import the resulting *DLL* file to a process or a batch, or you can use the built-in script editor and compiler to create your assemblies directly within client module setup windows.

Note: The .NET Code Module also enables you to run custom code during batch processing. It is a generic module whose purpose is to run custom code in a context that is independent from any other module. It provides a separate programming interface that does not directly share any components with the client-side scripting interface used by other modules. For details on configuring and running this module, see *OpenText Intelligent Capture - .NET Code Module Guide (ECPCORE-CNT)*.

7.1.2 Understanding How Scripts Are Associated With Modules

Scripts are run directly by client modules. Many modules have scripting interfaces that a script can implement to automate tasks associated with the module. For example, to automate optical character recognition (*OCR*), you would create a script that uses a NuanceOCR interface, then open the associated module in setup mode to configure it to run the script.

Not all modules have scripting interfaces. The "Programming reference" on page 452 section lists the available scripting namespaces and the modules with which they are associated.

7.2 Creating Scripts

This section explains how to create scripts that can be used with Intelligent Capture processes.

7.2.1 Choosing a Script Editing Environment

Because an Intelligent Capture script is a .NET assembly, you can use any .NET programming tool to create one. Intelligent Capture has been specifically designed and tested to work with scripts created from within the following environments:

- The Intelligent Capture built-in script editor

  When you set up a module, you can type your .NET code directly into a script editing window. A built-in compiler creates the final assembly automatically for you. The documentation for each module describes how to set up that module.

  If you use the built-in editor, there is no need to purchase, install, and maintain a separate application for creating your scripts. When you create a script in this editor, a structure that implements every event handler is created for you automatically, making it easier to just fill in code for the specific events you want to handle.

- Microsoft Visual Studio

  Visual Studio is the main programming tool for Windows .NET Framework projects. You might choose to use Visual Studio to take advantage of its full integrated development environment—which includes wizards, form designers, and other graphical programming aids; quick access to syntax reference information and detailed .NET documentation; and better debugging tools. Using Visual Studio is required if you want to create multi-language user interfaces in order support users in different locales. Finally, if the script developer will not be the person configuring modules to run the scripts, providing a precompiled *DLL* file to an administrator or a module operator simplifies the task of configuring the module and reduces the chance of a mistake occurring.

7.2.2 Setting Up a Client-Side Scripting Project

In programming terms, a client-side script is a class library. This is the project type to select if you create a script using Visual Studio.

Each script requires the following basic structural components:

- References to any external assemblies used in the script. These must usually include the following:
  - The common Intelligent Capture scripting interface (`Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`).
  - The module-specific scripting interface. Refer to the "Programming reference" on page 452 to learn which interfaces apply to which modules.

    Note: The Documentum Advanced Export module is the only module for which there are two module-specific assemblies that scripts need to reference. In addition, most scripts for this module will also need to reference the Documentum Foundation Classes (DFC) Primary Interop Assembly (`Documentum.Interop.DFC.dll`), which is not provided with Intelligent Capture but is part of the DFC itself. For more details, refer to the "Programming reference" on page 452 for the Documentum Advanced Export module.

  Assemblies should be referenced without path names. If you need to reference an assembly that is not an Intelligent Capture scripting assembly, either you must ensure that it is installed to the client machine's Global Assembly Cache (*GAC*), or you must add it as a scripting data file to the module. For details about adding data files, see"Using Data Files In a Script" on page 447 .

  Note: System assemblies should be available on every machine. The Microsoft .NET Framework installs them to the Global Assembly Cache.
- At least one class to contain the methods you want to attach to script events.
- One or more methods to run within client modules. These are your event handlers.

The programming languages supported by Intelligent Capture are Visual C# and Visual Basic. Here is a very simple Visual Basic "Hello, World!" client-side script that can be run with the ScanPlus module:

```
Imports Emc.InputAccel.QuickModule.ClientScriptingInterface
Imports Emc.InputAccel.ScanPlus.Scripting
Imports System.Windows.Forms

Public Class ScanPlusModuleEvents
    Sub ModuleStart(arg As IApplication)
        MessageBox.Show("Hello, world!")
    End Sub
End Class
```

Here is the same script written in Visual C#:

```
using Emc.InputAccel.QuickModule.ClientScriptingInterface;
using Emc.InputAccel.ScanPlus.Scripting;
using System.Windows.Forms;

public class ScanPlusModuleEvents
{
    public void ModuleStart(IApplication arg)
    {
        MessageBox.Show("Hello, world!");
    }
}
```

The assembly references for this script are the following:

- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `Emc.InputAccel.ScanPlus.ScriptingInterface.dll`
- `System.Windows.Forms.dll`

These references are stored in a separate file. In Visual Studio they are part of the project settings; in the built-in Intelligent Capture script editor they are listed in a separate window and can optionally be saved or retrieved from an *XML* document. The code and the references together make up the entire script.

When you configure the ScanPlus module to map the `ModuleStart` event to the `ModuleStart` method in this script, the module displays a message box as soon as it starts. Although in this sample both the method name and the event name are the same, the names do not have to match if you map events one by one. You can give your methods any names you want, but it is generally easier to maintain them if they are consistently named. The names *do* need to match if you want to make it easy to automatically map entire interfaces at once (for example, all of the module events). Refer to "Adding Automapping Information" on page 442 for information about enabling automapping within your scripts.

Of course you can create considerably more complex scripts than this one. Scripts can be created as text scripts using the built-in script editor, but for more complex scripts it is easier to use a full programming environment. If you decide to use the built-in editor, you can load and save your scripts from text files, which gives you a backup mechanism along with the option to use a more robust external text editor to edit them.

Note: The script editor is limited to 32767 characters and truncates accordingly.

7.2.3 Creating Event Handlers

An event handler is a script method that runs when a certain event occurs within a module. The method (and the class in which it is contained) must be declared as public, and it must be written to accept specific types of inputs and return a specific type of output as expected by the module for that event. Any method that matches this event signature becomes available for mapping within the module. The function and parameter names can be anything you want, as long as they have the expected data types.

Events fall into the following categories:

- **Module events**: These are process-independent events that always apply to the module regardless of which batch is being run. Examples are `ModuleStart` and `LostServerConnection`. Although you need a process that includes a module to set up that module, after the module events are configured for the first time, the mappings apply to that module when it is used in any process or batch on the same server. Most modules have module events, though they are not necessarily the same for all modules.
- **Task events**: These apply only to a specific process or batch, and vary from module to module depending on the types of tasks that the module performs. Examples of task events are `BeforeScan` and `AfterNodeProcessed`. As with other module settings, if you configure these mappings within a process, they apply to all batches based on that process. If you configure them within a batch, they apply to that batch only. Most modules have task events, though they are not necessarily the same for all modules.

Note: It is possible to apply task event mappings to all processes and batches on the server at the same time, by implementing the `IProvideTaskEventObject` interface. For details, see"Programming reference" on page 452.

The actual events available in each module are defined by that module. The names of the event interfaces (which contain the events) end with the text `Events`; for example, `ITaskEvents` or `IScanPlusModuleEvents`. If a specific module's namespace does not have an interface for either module events or task events, it uses the default interface for that set of events from the common scripting namespace (`Emc.InputAccel.QuickModule.ClientScriptingInterface`). These default interface names are `IDefaultModuleEvents` and `IDefaultTaskEvents`.

For detailed information about the interfaces used in each module, see"Programming reference" on page 452. You can also see the list of available events when you run a module in setup mode, which is where module events are mapped to script methods. You can embed information into a script that enables a user setting up the module to configure the mappings automatically. Refer to "Adding Automapping Information" on page 442 for details about adding automapping information to a script.

7.2.4 Adding Automapping Information

You can add metadata to a script that enables event mappings to be configured automatically. This automapping information is specified using .NET attributes. The available attributes define mappings at different levels:

- `AutoMappingClassAttribute`: This attribute is used on a class to map each method in the class to an event in a specific scripting interface. This lets you easily map all of the events, but provides limited flexibility because it is always a 1:1 mapping.
- `AutoMappingMethodAttribute`: This attribute is used to map individual methods to individual events. It lets you map only a subset of events in an interface, or to map multiple methods to the same event.

Note: Specifying mapping information within the script does not force it to be used within a process or a batch. When setting up a module with the script, a user can choose either to apply the automapping information provided in the script, or to map events manually.

Related Topics

"Creating Event Handlers" on page 441

"Method Mapping Rules" on page 442

"Creating Custom Script Parameters" on page 443

7.2.5 Method Mapping Rules

A method must meet the following criteria to be available for mapping:

- It must be part of a class that is declared as public.
- The method itself must be declared as public.
- Neither the method nor the class may be declared as abstract or generic.
- The method must have input and output data types compatible with the event you want to map it to, which means:
  - The event parameter types match, implement, or inherit the types defined by the method; and
  - The method return types match, implement, or inherit the types defined by the event.
- The method must accept a fixed number of arguments.
- The method must not be a constructor, a destructor, or a property getter or setter.
- The method must not be overloaded.
- The method must not be defined in the System.Object class.

- The method must not implement either the `IDisposable` interface or the `IProvideTaskEventObject` interface.

7.2.6 Creating Custom Script Parameters

Some scripting events accept custom parameters that can be configured when the events are mapped in setup mode. This enables you to create a more flexible script whose behavior can change without requiring a change to the script code itself.

You can create a custom parameter for any event whose handler provides access to a property that implements the `Emc.InputAccel.QuickModule.ClientScriptingInterface.ICustomParameter` interface. For example, the ScanPlus event handler for `AfterPageAdded` takes an `ITaskEventArgs` object as a parameter, and this object has a `CustomParameter` property that implements `ICustomParameter`.

To create a custom parameter for this event, you would do the following:

1. Create a class that defines your custom parameter variable names and types.

   This class does not have to be declared as public, but it must be marked with the `SerializableAttribute` attribute. It must also have a default constructor. (In .NET this constructor will be created automatically if you do not create one yourself.) Here is an example in Visual Basic:

```
Imports Emc.InputAccel.QuickModule.ClientScriptingInterface
Imports Emc.InputAccel.ScanPlus.Scripting
Imports System

<SerializableAttribute> _
Friend Class MyCustomParameters
    Private _CustomParam1 As String = "My Value"
    Property CustomParam1 As String
        Get
            Return _CustomParam1
        End Get
        Set(ByVal value As String)
            _CustomParam1 = value
        End Set
    End Property
End Class
```

2. Apply the `CustomParameterType` attribute to the event handler. To access the custom parameters, you must get the `Value` property of the `ICustomParameter` object. For the `AfterPageAdded` event, this might look like the following, which assigns the value of `CustomParam1` to a function variable named `customValue`:

```
Public Class ScanPlusTaskEvents
    <CustomParameterType(GetType(MyCustomParameters))> _
    Public Function AfterPageAdded(taskInfo As ITaskEventArg, pageNode As
IBatchNode) As ScriptResult
        Dim customEventData As MyCustomParameters = taskInfo.CustomParameter.Value
        ' IF PARAMETER HAS NOT BEEN SET, GET DEFAULT VALUE
        If( customEventData Is Nothing ) Then
            customEventData = New MyCustomParameters()
        End If
        Dim customValue As String = customEventData.CustomParam1
        Return ScriptResult.OK
    End Function
End Class
```

Notice that this code contains a check for the assignment of a non-null value to the custom parameter object (`customEventData`). It is possible that the value could be null if the event handler is assigned but never customized (the **Customize** button in setup mode is never clicked). The check ensures that in this case, the default value is assigned instead.

Note: When you use Visual C#, you must explicitly cast the `ICustomParameter.Value` property to your custom parameter type, as in this example:

```
MyCustomParameters customEventData = (MyCustomParameters)taskInfo.CustomParameter.Value;
```

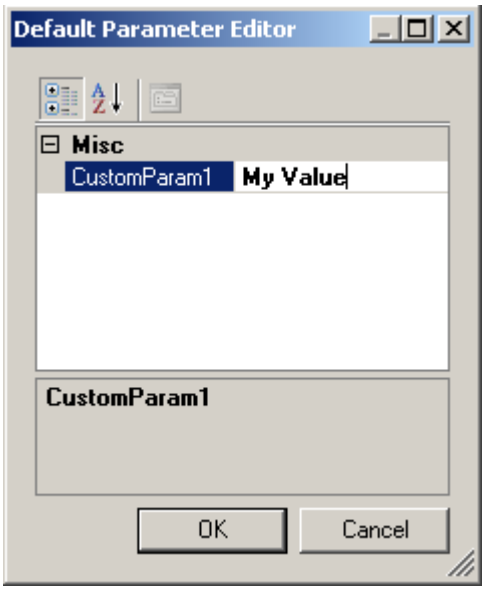
Here is the entire script in Visual C#:

```
using Emc.InputAccel.QuickModule.ClientScriptingInterface;
using Emc.InputAccel.ScanPlus.Scripting;
using System;

[SerializableAttribute]
internal class MyCustomParameters
{
    private string _CustomParam1 = "My Value";
    public string CustomParam1
    {
        get
        {
            return _CustomParam1;
        }
        set
        {
            _CustomParam1 = value;
        }
    }
}

public class ScanPlusTaskEvents
{
    [CustomParameterType(typeof(MyCustomParameters))]
    public ScriptResult AfterPageAdded(ITaskEventArg taskInfo, IBatchNode pageNode)
    {
        MyCustomParameters customEventData =
(MyCustomParameters)taskInfo.CustomParameter.Value;
        // IF PARAMETER HAS NOT BEEN SET, GET DEFAULT VALUE
        if( customEventData == null )
        {
            customEventData = new MyCustomParameters();
        }
        string customValue = customEventData.CustomParam1;
        return ScriptResult.OK;
    }
}
```

If you were to configure this event handler with a custom parameter, Intelligent Capture would display the built-in default parameter editor, which provides a basic interface for setting values of various data types.



The default parameter editor is an implementation of the .NET Framework `PropertyGrid` control. It is a generic window that is not optimized for any particular purpose. It is suitable for simple customizations but might exhibit a few quirks. For example, each value specified by the user is validated as soon as the edit field loses the input focus, so even if the user clicks **Cancel**, an invalid entry produces an error message.

For information about creating your own parameter editor, see "Creating a Replacement Parameter Editor" on page 446.

Customizing the Default Parameter Editor

You can customize the default parameter editor to some extent by applying attributes to the definitions of the parameters themselves. The attributes you can use are defined in the .NET Framework `System.ComponentModel` namespace. Some of the attributes that might be useful include the following:

- `CategoryAttribute`: Changes the category heading for the parameter. By default, all parameters are placed into a category named **Misc**.
- `DescriptionAttribute`: Provides a description of the parameter that the user can see in the editor window.
- `PasswordPropertyTextAttribute`: Masks the value of a parameter so that its actual characters do not appear on the screen.

Refer to the Microsoft Developer Network (MSDN) documentation for more details about these and other attributes defined in this namespace.

As an example, the following modification to the above Visual Basic definition of `CustomParam1` creates a description of this parameter that will be displayed at the bottom of the default editor when the user selects the parameter in the list:

```
<SerializableAttribute> _
Friend Class MyCustomParameters
    Private _CustomParam1 As String = "My Value"
    <System.ComponentModel.DescriptionAttribute("Sets the maximum number of pages
allowed in a document.")> _
    Property CustomParam1 As String
        Get
            Return _CustomParam1
        End Get
        Set(ByVal value As String)
            _CustomParam1 = value
        End Set
    End Property
End Class
```

Creating a Replacement Parameter Editor

You also have the option of providing your own custom parameter editor as a replacement for the built-in editor. You can do this as follows:

1. Create a custom editor class that implements the `Emc.InputAccel.QuickModule.ClientScriptingInterface.ICustomParameterEditor` interface. This interface contains a single method whose final parameter is a generic `Object` that you will need to cast to your custom parameter class type. Your editor code can then modify the parameter values and return the updated object.

   Here is an example of a simple Visual Basic custom editor that displays a form (`MyCustomEditorWindow`) and returns the value input by the user. The code for the form itself is not shown here. Because it is much easier to create forms in a visual editor, you would most likely want to create this project using Visual Studio rather than the script editor built into Intelligent Capture.

   ```
   Friend Class MyParameterEditor
       Implements ICustomParameterEditor
       Public Function EditParameter(parentWindow As IWin32Window, workflowStep As
   IWorkflowStep,
         objectToEdit As Object) As Object Implements
   ICustomParameterEditor.EditParameter
           Dim updatedParams As MyCustomParameters = objectToEdit
           MyCustomEditorWindow.ShowDialog()
           updatedParams.CustomParam1 = MyCustomEditorWindow.TextBox1.Text
           Return updatedParams
       End Function
   End Class
   ```

   Here is the same custom editor in Visual C#:

   ```
   internal class MyParameterEditor : ICustomParameterEditor
   {
       public Object EditParameter(IWin32Window parentWindow, IWorkflowStep
   workflowStep, object objectToEdit)
       {
           MyCustomParameters updatedParams = (MyCustomParameters)objectToEdit;
           MyCustomEditorWindow.ShowDialog();
           updatedParams.CustomParam1 = MyCustomEditorWindow.TextBox1.Text;
           return updatedParams;
       }
   }
   ```

2. Apply the `CustomParameterEditor` attribute to the class that defines your custom parameters. This attribute takes a parameter that indicates the type of the editor class itself. Here is how it might look in Visual Basic:

```
<SerializableAttribute> _
<CustomParameterEditor(GetType(MyParameterEditor))> _
Friend Class MyCustomParameters
    ' PARAMETER DEFINITIONS GO HERE
End Class
```

Here it is in Visual C#:

```
[SerializableAttribute]
[CustomParameterEditor(typeof(MyParameterEditor))]
internal class MyCustomParameters
{
    // PARAMETER DEFINITIONS GO HERE
}
```

7.2.7 Using Data Files In a Script

You can provide data files and access them from within your event handlers. A data file can be of any file type. When a module is configured to run the script, the data file can be added as well.

An uploaded data file gets stored at a location relative to the compiled script *DLL*. If you specify only a file name, it is placed in the same folder as the script. You can create a hierarchy beneath the script folder by specifying a relative path when you upload the data file. For example, if you specify `data\datafile.txt` as the file name, then a folder named `data` is created automatically on the server, and the file is placed within it.

Note: When uploading a file, you can change its name and location as desired. Its name on the server does not have to match the original name of the file you are uploading.

During execution, the script's working folder is not necessarily set to its own location, so your script will need to retrieve the location and use it to determine the path to the data file. The following Visual Basic example does this by using the .NET Framework `GetExecutingAssembly` method to find a data file named `datafile.txt` that's stored in the same folder as the script:

```
Dim fileReader As New
StreamReader(Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
, "datafile.txt"))
Dim firstLine = fileReader.ReadLine()
```

Here is the Visual C# equivalent:

```
StreamReader fileReader = new
StreamReader(Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)
, "datafile.txt"));
string firstLine = fileReader.ReadLine();
```

To use this sample, you must add namespace references to `System.IO` and `System.Reflection`.

If your script references assemblies other than system assemblies and Intelligent Capture scripting assemblies, you should add those referenced assemblies as data files to ensure that they are found by the script at run-time.

7.2.8 Creating a User Interface

You can allow batch operators to interact with a client-side script. To create a simple user interface, it might be sufficient to use built-in .NET Framework features. For example, you can give the user a choice between two options simply by displaying a window with a "yes-or-no" question, which can be done with methods from the built-in `System.Windows.Forms.MessageBox` class. Here is a simple example in Visual Basic:

```
Imports Emc.InputAccel.ScanPlus.Scripting
Imports Emc.InputAccel.QuickModule.ClientScriptingInterface
Imports System.Windows.Forms

Public Class ScanPlusTaskEvents

    Public Function BeforeBatchClose(ByVal taskInfo As ITaskEventArg, ByVal
autoBatchCreation As Boolean,
      ByRef closeMode As BatchCloseMode) As ScriptResult
        Dim SendEmail As DialogResult
        SendEmail = MessageBox.Show( "Do you want to send an email notification that the
batch is
      ready for further processing?", "Send Notification", MessageBoxButtons.YesNo )
        If( SendEmail = DialogResult.Yes )
            ' INSERT CODE TO SEND EMAIL HERE
        End If
    End Function

End Class
```

To create a more sophisticated user interface, you can write your script in Microsoft Visual Studio. The built-in Windows Forms Designer makes it easy to add standard user interface controls such as text boxes and radio buttons. "Choosing a Script Editing Environment" on page 438 has additional information about the advantages of using Visual Studio to create your scripts.

Creating Locale-Specific User Interfaces

If you use Visual Studio to create a client-side script, you can create your user interface in multiple languages. When the script runs, it appears in the same language as the module itself.

The documentation for Visual Studio contains information about localizing project resources into multiple languages. You can use this information to create localized scripts, but be aware of the following points:

- You *must* digitally sign your assembly with a strong name to use localized resources in a script.
- When you compile your project, the resources for each language are saved in a separate assembly file. This satellite assembly is placed in a sub-folder named with a locale identifier (for example, `de-de\Script.resources.dll` for a German resource file). When you add the compiled script to a module, you must add each satellite assembly to the module using the same relative path output by the compiler.

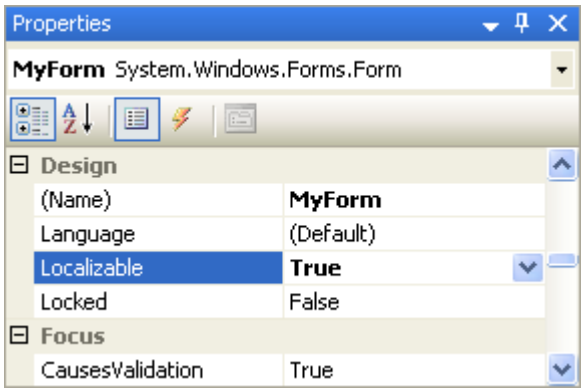
Here are the main steps you would use to create a script that can run in either English or French:

1. With English configured as your system default language, use Visual Studio to create a script for a module, as explained in "Setting Up a Client-Side Scripting Project" on page 439.

2. Create one or more forms.

   In Visual Studio 2005, choose **Project > Add Windows Form** to create a form.

3. For each form, set the **Localizable** property to **True**.



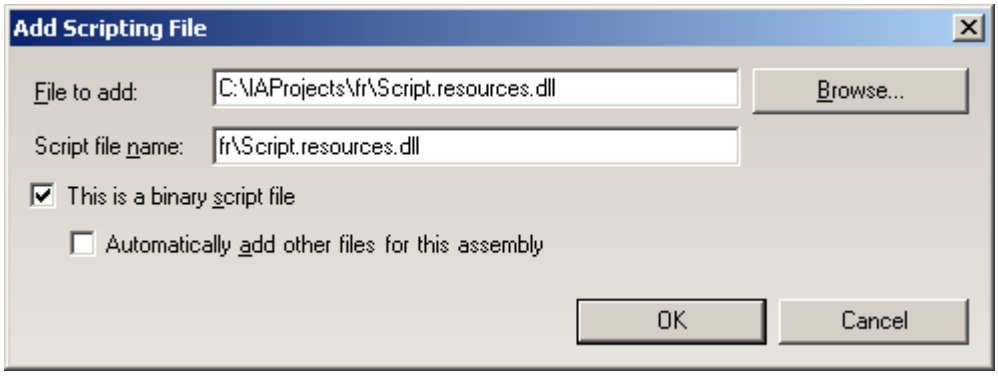
4. For each form, set the **Language** property to **French**, then edit the form so that the text is in French. You can also reposition controls if necessary, and the changes apply only to the French version of the form.

5. Create one or more event handlers that will run when the scripting is configured for the module. Add code to the event handlers to display the forms you created, by using the `ShowDialog` method for example.

6. Digitally sign the assembly in the project properties (**Signing** option). Select **Sign the assembly** and specify a file name. Setting a password is optional.

7. Build the project.

8. When the module is configured to run the script, the satellite assemblies containing the localized resources must be imported along with the main assembly. For example, if the project is named *<Script>*, the `Script.resources.dll` file should be uploaded as `fr\Script.resources.dll`.



7.2.9 Using the Sample Scripts

Each module with a scripting interface also includes one or more sample scripts. You can find the scripting samples in the program files folder, under *<drive>*:\Program Files\InputAccel\Client\src\Sample Capture System\ScriptSource\Client-side Scripts.

Each sample is a complete Visual C# project. You can view and edit the source files directly, and you can compile each sample to run it within a process or a batch.

There are several options available for compiling a sample script:

- Open the project file in Visual Studio and build it.

  This is the simplest method, and the recommended one if you have Visual Studio available. Within each sample folder, the project file has the extension CSPROJ.

- Use the .NET command line compiler (MSBuild.exe).

  This is a free tool that gets installed with .NET Framework. It is located in the Microsoft.NET\Framework\<<*version*>>\ sub-folder of the Windows folder. To use it, provide the full project file path as a parameter to the command; for example:

  ```
  msbuild "<drive>:\Program Files\InputAccel\Client\src\Sample Capture System\ScriptSource\Client-side Scripts\DocumentumAdvancedExport\VirtualDocument\VirtualDocument.csproj"
  ```

- Copy the source code into the built-in Intelligent Capture script editor, and compile it.

  This is the most complicated method, but you can do it if necessary by following these steps:

  1. Run the module for setup.
  2. Select the **Scripting** option from the navigation panel.
  3. Click the **Manage script files** button to display the **Manage Scripts** window.
  4. Click the **Create text script** button to display the **Text Scripting File** window.
  5. From the **Language** list box, select **C#**.
  6. Copy and paste the entire contents of each *CS* file in the project into the script editor window. The *CS* files can be pasted in any order. Paste the contents of one *CS* file after all the ones before it, *except* that all of the `using` statements at the top of each *CS* file need to be consolidated at the top of the script. You can eliminate duplicate `using` statements if you want to, but it is not necessary to do so.

7. Click **References** to display the **Add Reference** window. Then open the `CSPROJ` file in a text editor and look for *XML* `<Reference>` tags. Add the assembly from each reference `Include` attribute into the reference window by checking the appropriate box, or browsing to the file if it does not appear in the list.

8. Click **Compile** to compile the .NET code, automatically creating the final assembly.

7.2.10 Programming Tips

In general, programming guidelines that apply to writing good .NET programs also apply to writing client-side scripts for Intelligent Capture. You can find many good tips and examples on the Microsoft website (http://www.microsoft.com/).

One feature of .NET programming that is slightly different for scripts is that an object is not guaranteed to be static throughout the lifetime of a script. Therefore, to test the equality of objects, it is often more reliable to test the equality of object properties that are known to be unique. For example, rather than comparing `IBatchInformation` objects directly, you can compare their Id properties. Alternatively, if you need to keep a reference to a particular object for its entire usable lifetime, you can assign it to a static class variable, as in the following example for Documentum Advanced Export:

```
public class DocumentumExportTaskEvents : IDocumentumExportTaskEvents
{

    static IBatchInformation currentBatch;

    public void BeforeExportRun(IBeginTaskEventArgs taskInfo)
    {
        currentBatch = taskInfo.Task.Batch;
    }

}
```

You could then use the variable *<currentBatch>* in any other task event.

Here is the same example in Visual Basic:

```
Public Class DocumentumExportTaskEvents
    Implements IDocumentumExportTaskEvents

    Shared currentBatch As IBatchInformation

    Public Sub BeforeExportRun(ByVal taskInfo As IBeginTaskEventArgs) Implements
IDocumentumExportTaskEvents.BeforeExportRun
        currentBatch = taskInfo.Task.Batch
    End Sub

End Class
```

Note: Each object has a valid scope, and the static variable does not provide access to the object beyond that scope. In the example, the Task object (and therefore the associated Batch object) becomes invalid when the task is completely processed.

7.3 Running scripts

After a script is created, it can be added to a process step or a batch step by running the appropriate client module in setup mode. Each module's documentation contains information about setting up scripting for the module, which involves mapping events to script methods and then setting the values of any custom parameters.

Mapped script methods run only in production mode. After a method is mapped to an event, it runs automatically when the event occurs. In most cases, module operators do not have to do anything more than they would normally do when running their batches.

7.4 Programming reference

This section contains reference information for the available scripting .NET interfaces.

7.4.1 Namespaces

This section contains syntax and usage information for each available Intelligent Capture scripting library.

- Emc.InputAccel.DocumentumExport Namespace
- Emc.InputAccel.DocumentumExport.Scripting Namespace
- Emc.InputAccel.DocumentumExport.Setup Namespace
- Emc.InputAccel.NuanceOCR.Scripting Namespace
- Emc.InputAccel.QuickModule.ClientScriptingInterface Namespace
- Emc.InputAccel.ScanPlus.Scripting Namespace
- Emc.InputAccel.WebServicesScripting Namespace

7.4.2 Samples

This section contains information about accessing and using the sample scripts provided with Intelligent Capture.

Each module with a scripting interface also includes one or more sample scripts. Each sample is a complete Visual Studio C# project. It is recommended that you use Visual Studio to compile and use the samples, but you can also view, modify or recompile a sample script from within Intelligent Capture. You can find the scripting samples in the program files folder, under *<drive>*:\Program Files\InputAccel\Client\src\Sample Capture System\ScriptSource\Client-side Scripts.

7.4.2.1 Documentum Advanced Export Scripting Samples

This section contains information about the Documentum Advanced Export client-side scripting samples that are provided with the module.

7.4.2.1.1 Documentum Advanced Export Scripting Sample: AdvancedRescan

The `AdvancedRescan` sample script enables the manual triggering of a Documentum process. It also creates a new minor version of a `dm_document` object, and then triggers the process containing the object to continue to the next step.

Once Documentum Advanced Export updates the existing `dm_document` object on the Documentum side, it can resume the process. This step enables creation of a single Documentum process that handles all of the document rescan logic.

Overview

The `AdvancedRescan` script sample uses the Web Services Input module to make a web call for rescan. Manual activity for completion happens after the web call. The process must contain the following modules as steps:

- Web Services Input (`wsinput_rescan.mdf`)
- RescanPlus (`RescanPlus.mdf`)
- Documentum Advanced Export (`iaexdm.mdf`)

The *Web Services Guide* contains information on how to set up and use the Web Services Input module.

The sample exports rescanned pages to the original document: it applies version information, replaces the document in the package with the rescanned version, and completes the manual activity. This process enables the Documentum process to flow without a break.

References

The assembly references for this sample are:

- `Emc.InputAccel.DocumentumAdvancedExport.dll`
- `Emc.InputAccel.DocumentumAdvancedExport.Setup.dll`
- `Emc.InputAccel.DocumentumAdvancedExport.Scripting.dll`
- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `System.dll`

- System.Collections.Generic.dll

Description

The BeforeExportRun event handler sets flags indicating that the module runs in rescan mode based on settings in the export list. Throughout the script, other event handlers access the global variables *_rescanSettings*, *_isRescanModeTask*, and *_curTask*.

```
public void BeforeExportRun(IBeginTaskEventArgs taskInfo)
{
  _rescanSettings = taskInfo.RescanSettings;
  _isRescanModeTask = taskInfo.IsRescanModeTask;
  taskInfo.ProcessRescanModeWorkflowManual = true;
  _curTask = taskInfo.Task;
}
```

The CaptureFlow used for this sample is called WSInputRescan.XPP, and is located in the sample projects folder. The WS Input step uses wsinput_rescan.mdf to receive rescan settings from the Documentum Advanced Export step, and pass these settings to the RescanPlus step.

The script then defines the AfterExportListLoad event handler. This event handler finds and modifies version export list settings and enables version support for the rescan mode task. It then creates a new minor version of the Documentum object.

The AfterDocumentObjectProcessed event handler checks the *_isRescanMode* flag and determines if the Documentum object is running in rescan mode. If the object has been rescanned, the script finds the existing business process and gets the process ID. It replaces the old object with the new one, and completes the task.

7.4.2.1.2 Documentum Advanced Export Scripting Sample: BOF

The BOF scripting sample demonstrates how to execute methods of a Documentum service business object. The BOF allows developers to design custom Documentum object extensions.

Overview

This script gives the service business object a Documentum repository name and dm_folder id, and then the service business object clears the folder contents. The sample calls this service business object for each folder defined for the Documentum Advanced Export step.

This sample requires compiling and installing a service business object into Documentum. The service business object source files are located in the script sample folder BOF\Java. The Documentum documentation contains more information on business objects and service business objects.

For setup in Documentum Advanced Export, the **All definitions** list must include a dm_folder object containing a dm_document object. Subsequent exports must go to the same folder to ensure that its contents are cleared before each export.

References

The assembly references for this sample are:

- `Emc.InputAccel.DocumentumAdvancedExport.dll`
- `Emc.InputAccel.DocumentumAdvancedExport.Scripting.dll`
- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `System.dll`
- `System.Windows.Forms.dll`

Description

Within `BeforeNonDocumentObjectProcessed`, the script declares a `BeforeNonDocumentObjectProcessed` method.

The script then checks for the Documentum object to determine if it is a `dm_folder` supertype. If the object supertype is `dm_folder`, then an *SBO* method is called from the Documentum object being processed, and the object is cleared before export.

```
public class BOFScript
    {
        // Repository name where folder should be deleted
        private string repository;

        // Save repository name for future use
        [AutoMappingMethod(typeof(IDocumentumExportTaskEvents),
            Method = "AfterThirdPartyConnectionEstablished")]
        public void AfterThirdPartyConnectionEstablished(
            IAfterConnectionEstablishedEventArgs connection)
        {
            // Save repository name...
            repository = connection.RepositoryName;
        }

        // Implements general functionality for the class
        [AutoMappingMethod(typeof(IDocumentumExportTaskEvents), Method =
"BeforeNonDocumentObjectProcessed")]
        public void BeforeNonDocumentObjectProcessed(INonDocumentConveyorItem objectInfo)
        {
            if (objectInfo.Definition.Object.SuperType == SetupConstants.folder)
            {
                object result =
objectInfo.BOFAccessor.InvokeSBOMethod("com.emc.ia.ICleanUpFolderService",
                                                                    "CleanUpFolder",
                                                                    new object[]
{ repository, objectInfo.ObjectId});
            }
        }
    }
```

7.4.2.1.3 Documentum Advanced Export Scripting Sample: VirtualDocument

Use a virtual document as a container for other documents of multiple types in a hierarchical structure. The `VirtualDocument` sample collects objects being exported as virtual documents. The Documentum documentation contains more information on virtual documents.

A recommendation is to set up the Documentum Advanced Export step to export several documents and folders per task. This scripting sample requires installing an *SBO* in the Documentum repository to create the virtual document. No unique CaptureFlow settings are required to run this sample.

References

The assembly references for this sample are:

- `Emc.InputAccel.DocumentumAdvancedExport.Scripting.dll`
- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `System.dll`
- `System.Collections.Generic.dll`
- `System.Collections.Specialized.dll`

Description

The sample first declares the `DocumentumExportTaskEventHandler` class. It initiates the `List` member (*`docs`*) to append a list of document object ids from Documentum Advanced Export. It then initiates the `IBOFAccessor` member (*`bof`*) to call *SBO*s. It finally initiates a string (*`repository`*) representing the name of the Documentum repository where to save the virtual document.

```
public class DocumentumExportTaskEventHandler
    {
        // List of document IDs to be added to virtual document
        private List<string> docs;

        // IBOFAccessor reference to call SBO
        private IBOFAccessor bof;

        // Repository name to save virtual document to
        private string repository;
```

The events defined by the `IDocumentumExportTaskEvents` interface are: `AfterThirdPartyConnectionEstablished`, `AfterTaskProcessed`, and `AfterDocumentObjectProcessed`. Because each class is declared as an AutoMappingClass, its methods can be mapped automatically to the corresponding events in the Documentum Advanced Export scripting interface.

The `AfterThirdPartyConnectionEstablished` handler then saves the repository name (*`repository`*) and the *BOF* accessor (*`bof`*) for export.

```
public void AfterThirdPartyConnectionEstablished(
            IAfterConnectionEstablishedEventArgs connection)
        {
```

```
            // Save repository name...
            repository = connection.RepositoryName;

            // ... and BOFAccessor
            bof = connection.BOFAccessor;
        }
```

The `AfterDocumentObjectProcessed` class adds each exported document to the list of documents (*doc*) to be attached to the virtual document.

```
public void AfterDocumentObjectProcessed(IDocumentConveyorItem objectInfo)
        {
            // Create a list if it is not created yet
            if(docs == null)
            {
                docs = new List<string>();
            }

            // Add document to list
            docs.Add(objectInfo.ObjectId);
        }
```

The `AfterTaskProcessed` class calls the *SBO* method to create the virtual document. It then populates it with the exported documents, appending the exported simple document to the new virtual document object. Finally, the class cleans up any remaining resources, and saves the virtual document.

```
public void AfterTaskProcessed(IFinishTaskEventArgs taskInfo)
        {
            // Call SBO method - it creates virtual document and
            // populates it with the supplied list of documents
            bof.InvokeSBOMethod("com.emc.captiva.ia.sbo.IVirtualDocService",
                "createVirtualDoc",
                new object[] { repository, docs.ToArray()} );

            // Clear list
            docs.Clear();
        }
```

7.4.2.2 NuanceOCR Scripting Samples

7.4.2.2.1 NuanceOCR Scripting Sample: ScriptNuanceOCR

The `ScriptNuanceOCR` scripting sample demonstrates how the NuanceOCR module uses message boxes and log files to display important information about the *OCR* process.

References

The assembly references for this sample are:

- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `Emc.InputAccel.QuickModule.Plugins.Script.dll`
- `NuanceOCR.Scripting.dll`
- `System.dll`
- `System.Windows.Forms.dll`

Description

When the `AfterNodeProcessed` event is fired, NuanceOCR populates the processing log with the following:

- Parameter values
- Error message information
- *OCR* engine settings
- A detailed list of results for each zone and each page.

A button on the bottom of the NuanceOCR main window enables exporting this log to a file.

When the `AfterTaskProcessed` event is fired, NuanceOCR populates the processing log with the following:

- Parameter values
- Error message information
- Page result statistics
- Detailed information about the output files.

A button on the bottom of the NuanceOCR main window enables exporting this log to a file.

The last section of the scripting sample maps methods in the script to standard module events, using the `IDefaultModuleEvents` interface from the Emc.InputAccel.QuickModule.ClientScriptingInterface Namespace. It then displays a message box each time one of these module events fires. The message box shows that a module has started or that the connection to the server has changed. It can also display a more detailed reporting of error messages.

7.4.2.3 ScanPlus and RescanPlus Scripting Samples

ScanPlus and RescanPlus scripting interfaces are provided as .NET libraries. Scripts are written using standard .NET programming languages such as VB. NET (Visual Basic) and C#. These ScanPlus samples are also applicable to the RescanPlus module.

The related API references are as follows:

- Emc.InputAccel.ScanPlus.Scripting Namespace
- Emc.InputAccel.QuickModule.ClientScriptingInterface Namespace

Note: Note that RescanPlus uses the same scripting libraries as ScanPlus. The names of the namespaces and members use “ScanPlus” as their root.

IAScan Callback Functionality in ScanPlus Scripting

The Intelligent Capture 5.3 Scan module allowed customers to connect to a customer-supplied DLL, `Scan Callback DLL`, which enabled custom processes to

take place at various points within the scan loop. The following table provides descriptions of equivalent functionality for the `Scan Callback DLL` in ScanPlus.

Intelligent Capture 5.3 Scan Callback Function	Description	ScanPlus Client-Side Scripting Element	Description
`SetUpDLL( )`	Called when the user clicks the **DLL Setup** button in **Miscellaneous** tab of the **Scan Setup** window.	ICustomParameter Interface	Passes custom parameters to event handlers. Custom parameters can be used to create custom UI.
		ICustomParameterEditor Interface	Displays a custom dialog.
		`BeforeNewBatch` Method	Handles the event that is fired before a new batch is created. This method is used to modify or verify batch creation parameters.
`SetInstanceValue( )`	Called after a level 7, or batch, node has been created enabling the scan callback *DLL* to set the image address and endorser values.	`BeforeScan` Method	Handles the event that is fired before the scan operation begins. This method is used to set scan settings and includes parameters that can be used to get/set an IA Value.
		`ScanConfigurationSelected` Method	Handles the event that is fired when a user selects a scan configuration.
`SetInstanceValueError( )`	Called if an error occurs while Scan attempts to set an instance value supplied by the `SetInstanceValue( )` function.	There is no equivalent ScanPlus Client-Side Scripting task or module event.	Set/get IA Values to catch and process errors.
`PreLevelCreate( )`	Called immediately before Scan creates a new level in the tree.	`BeforeNewLevel` Method	Handles the event that is fired before a new node is added during scanning or importing.

Intelligent Capture 5.3 Scan Callback Function	Description	ScanPlus Client-Side Scripting Element	Description
`LevelCreateError( )`	Called only if an error occurs while Scan is attempting to create a level 7, or batch.	`BatchCreationError` Method	Handles the event that is fired when a batch creation error occurs.
`PostLevelCreate( )`	Called immediately after Scan creates a new node, but before it creates any child nodes.	`AfterNewLevel` Method	Handles the event that is fired when a new node is added.
		`AfterPageAdded` Method	Handles the event that is fired when a page node is added to the batch.
		`PrepareTask` Method	Handles the event that is fired after a batch is created, but before ScanPlus verifies the scan configuration of the batch. Use this method with `BeforeNewLevel` for functionality equivalent to `PostLevelCreate( )` at level 7.
`LevelFinish( )`	Called after Scan has created all child nodes of the specified level. If the level is 7, the function is called after Scan closes the batch.	`LevelFinished` Method	Handles the event that is fired when a scanner event creates the next sibling node for the current node. For functionality equivalent to `LevelFinish( )` at level 7, add a .NET Code Module step to the process after running ScanPlus to notify that ScanPlus closed the batch.

Intelligent Capture 5.3 Scan Callback Function	Description	ScanPlus Client-Side Scripting Element	Description
`BatchClose( )`	Called after the user invokes the **Close Batch** command but before Scan actually closes the batch.	`BeforeBatchClose` Method	Handles the event that is fired before a batch is closed. Use this method to display a dialog to the user before actually closing the batch.

7.4.2.3.1 BatchDivider Scripting Sample

The `BatchDivider` scripting sample demonstrates how to divide batches into predefined sizes.

References

The assembly references for this sample are:

- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `Emc.InputAccel.ScanPlus.ScriptingInterface.dll`
- `System.dll`
- `System.Data.dll`
- `System.Xml.dll`

Description

The `PrepareTask` method begins a try-catch loop by loading an *XML* document, called `BatchStructure.xml` (not shown in this code sample), then reading through its nodes and assigning two data values: *`nfolders_`* and *`ndocs_`*. If this try fails, then a default value is assigned for each. These values are used by subsequent methods to determine the size of the divided batches. The code sample shown here demonstrates how these data values are assigned:

```
XmlDocument doc = new XmlDocument();
doc.Load(path);

XmlNodeList nodes = doc.GetElementsByTagName("batch_divider");
foreach (XmlNode node in nodes)
{
    nfolders_ = int.Parse(node.Attributes[NFOLDERS_ATTR_NAME].Value);
    ndocs_    = int.Parse(node.Attributes[NDOCUMENTS_ATTR_NAME].Value);
    break;
}
```

The `BeforeNewLevel` method checks the current number of documents and folders, then creates a new batch or a new level when the predefined limits have been reached.

```
public ScriptResult BeforeNewLevel(ITaskEventArg taskInfo, IBatchNode parentNode,
IBatchNode previousNode, ILevelCreationParameters levelCreateInformation, ref
EventAction action, ref bool discard)
{
    ScriptResult sr = ScriptResult.OK;

    if (parentNode.Level == 3)
    {
        if (previousNode != null && previousNode.ChildCount() < ndocs_)
        {
            sr = ScriptResult.Cancel;
        }
        else if (taskInfo.Application.CurrentTask.BatchProcess.Tree.ChildCount(2) >=
nfolders_)
        {
            action = EventAction.NewBatch;
        }
    }
    else if (parentNode.Level == 2)
    {
        if (parentNode.ChildCount() >= ndocs_)
        {
            action = EventAction.NewLevel2;
            sr = BeforeNewLevel(taskInfo, parentNode.Parent(), parentNode,
levelCreateInformation, ref action, ref discard);
        }
    }
    return sr;
}
```

The `BeforeBatchClose` method checks the current number of child documents in the batch, and only closes the batch when the number of child documents is less than the number of allowed child documents in the last batch.

```
public ScriptResult BeforeBatchClose(ITaskEventArg taskInfo, bool autoBatchCreation, ref
BatchCloseMode closeMode)
{
    int completed = 0;
    if (taskInfo.Application.CurrentTask.BatchProcess.Tree.ChildCount(2) >= nfolders_)
    {
        completed = 1;
        foreach (IBatchNode node in
taskInfo.Application.CurrentTask.BatchProcess.Tree.Children(nfolders_))
        {
            if (node.ChildCount() < ndocs_)
            {
                completed = 0;
                break;
            }
        }
    }
    taskInfo.Application.CurrentTask.Value().SetInt("Completed", completed);
    return ScriptResult.OK;
}
```

ScanPlus Setup Requirements

Using the `BatchDivider` scripting sample requires configuring ScanPlus in a specific manner:

- ScanPlus trigger level: Any
- **Levels** window: Enable levels **0 (Page)**, **1 (Document)**, **2 (Folder)**
- **Event Actions** window: Define a single scanner event as follows:

  - **Scanner Events**: **Every N pages**
  - **Options**: 3
  - **Actions**: **New Document**
  - **Discard page**: not selected
- **Auto Batch Creation** window: Configure the **Batch name schema** and **Process schema** fields to automatically create uniquely-named batches from a specified process each time the script creates a new batch. *OpenText Intelligent Capture - ScanPlus Guide (ECPCORE-CSC)* explains how to configure these fields.

7.4.2.3.2 BatchFilter Scripting Sample

The `BatchFilter` scripting sample filters the batch list to accept and display only those batches where the ScanPlus steps have been completed. These are the batches in which the IA Value called <*Complete*> either is equal to `0` or is undefined.

References

The assembly references for this sample are:

- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `Emc.InputAccel.ScanPlus.ScriptingInterface.dll`
- `System.dll`
- `System.Data.dll`
- `System.Xml.dll`

Description

The `FilterBatchList` method only returns true when all ScanPlus steps have completed.

```
public bool FilterBatchList(IBatchInformation batch)
{
    bool accept = false;
    foreach (IWorkflowStep ws in batch.WorkFlowSteps)
    {
        if (string.Compare(ws.ModuleName, module_name_, true) == 0 &&
ws.Value().GetInt("Completed", 0) == 0)
        {
            accept = true;
            break;
        }
    }
    return accept;
}
```

7.4.2.3.3 EventMonitor Scripting Sample

The `EventMonitor` scripting sample demonstrates the implementation of events by printing important task event settings to a separate window each time an event is fired.

References

The assembly references for this sample are:

- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `Emc.InputAccel.ScanPlus.ScriptingInterface.dll`
- `System.dll`
- `System.Data.dll`
- `System.Drawing.dll`
- `System.Web.dll`
- `System.Web.Services.dll`
- `System.Windows.Forms.dll`
- `System.Xml.dll`

Description

When the methods in this script are mapped to task events in ScanPlus for Setup, each method reports a description of the current parameters at the time the event is fired. For every fired event, the **Event Monitor** window displays the time, type, instance name, batch name, parameter and parameter value. The `Add` method that is used to add information to the monitor window is defined in the file, `Report.cs`, included with this sample. Specifically, this method connects the firing of the event with the update of the display.

7.4.2.4 Web Services Scripting Samples

7.4.2.4.1 Web Services Scripting Sample: WS Input Rescan

The `WS Input Rescan` scripting sample demonstrates how to implement InputAccel 5.3 Rescan functionality using the Web Services framework without changing the existing *WSDL* interface. To successfully export data to Documentum, you must have installed the Documentum Advanced Export module.

Notes

- It is recommended that, inIntelligent Capture Designer, you do not open this sample together with other CaptureFlows. If you open this sample and then subsequently open other CaptureFlows, then the **Web Services Input Rescan** module is displayed in **Create Batch From** list of those CaptureFlows. To clear the **Web Services Input Rescan** module from the **Create Batch From** list, restart Intelligent Capture Designer.

- When running this sample in the RescanPlus module, select **Run Selected** instead of **Open Batch**.

References

The assembly references for this sample are:

- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `System.dll`
- `System.Data.dll`
- `System.Xml.dll`
- `WebServicesScripting.dll`

Description

This scripting sample demonstrates the improved flexibility of the Web Services subsystem. Because the WS Input module can only map simple *WSDL* parameters, the client-side scripting functionality is responsible for parsing the more complex parameters, like *`client_info_XML`* and *`pages`*.

This script handles the *`MappingDone`* event at level 7. The script creates batch tree nodes for levels 6 through 0, and adds images at level 0. The *XML* content of the *`client_info_XML`* parameter is parsed and written to batch level IA Values, preserving the tag names:

- *<WorkflowID>*
- *<PackageName>*
- *<ActivityName>*
- *<InputPortName>*
- *<UpdateContentType>*

The CaptureFlow for this sample is called `WSInputRescan.XPP`, and is located in the sample projects folder. The WS Input step uses `wsinput_rescan.mdf` to receive rescan settings from the Documentum process, and pass these settings to the RescanPlus step.

To successfully run this sample script, add a web service and hosting from which you can call the WS Input module, install the sample process, and then configure each module in the process. When adding a new web service for this sample, you should select and parse `IARescanWS.wsdl` to locate and register the corresponding service, called "IARescanWS". For specific instructions on adding a web service, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

After you have registered the web service for this sample, add a new Web Services Hosting. When registering the new hosting for this sample, enter `localhost` for the **Hosting name**, and leave the port number blank. For specific instructions on adding a new hosting, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

After adding a new Web Services Hosting, install the sample process using Intelligent Capture Administrator. For more information on the steps necessary to install a process, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

To set up the module steps in the WS Input sample:

1. Open the step settings for the WS Input step.
2. Select the **WS Input** pane from the left side of the window to display the **Mapping** information.
3. From the **Service for mapping** drop-down list, select the `IARescanWS` service. This should also populate the **Mapped method** drop-down list with the corresponding method, called `Rescan`.

   Note: Because the script will handle all mappings, it is not necessary to set any specific parameter mappings for this sample script.
4. In the **Batch Name Schema** field, set the batch naming rules for this sample, for example, “rescan-@(Now)”.
5. Select the **Scripting** pane from the left side of the window to display the **Scripting Settings** for the WS Input module.
6. Click **Manage script files**, then **Add file...**, then Browse to select the prebuilt scripting file named `Rescan.dll`.
7. Check the box labeled **This is a binary script file**, then click **OK**.
8. Verify that `Rescan.dll` is now listed in the **Manage Scripts** window, then click **OK**.
9. In the **Task event mappings**, select `MappingDone` and map it to the `RescanSample.WSInputTaskEvents.MappingDone` method. Click **OK** to save your changes and move onto the next module in the process.
10. Do not make any changes to the Scan or Rescan steps in this process.

The *Setup* section of the *Documentum Advanced Export Guide* contains more information on configuring Documentum. The *Documentum Advanced Export scripting sample: AdvancedRescan* section provides an overview of the Documentum Advanced Export module functionality in this sample script.

7.4.2.4.2 Web Services Scripting Sample: WS Output

The WS Output scripting sample demonstrates how to populate *SOAP* fields with client-side scripting, as well as how to verify that IA Values have been correctly populated. This sample script fills an array parameter of the request, where each array element is a structure with two fields, a string containing the *PDA XML* and a string containing the *URL* of the image.

References

The assembly references for this sample are:

- `Emc.InputAccel.QuickModule.ClientScriptingInterface.dll`
- `System.dll`
- `System.Data.dll`
- `System.Xml.dll`
- `WebServicesScripting.dll`

Description

This sample uses `WSOutputScan.xpp` and `ExportImagesWithZones.wsdl`. Both files are located in the sample projects folder.

Each step in this sample is triggered at level 1. The process starts after ScanPlus scans several pages. The *OCR* module then creates *XML* PDA for each page and Standard Export sends these pages to a network share. The WS Output module calls the Web Service (one call per document) and sends the *`Operator`* name and an array of pages (including the page image *URI* and *XML PDA* for each page) populated by the script.

To successfully run this sample script, add a web service, add a hosting, then configure each module in the process. When adding a new web service for this sample, select and parse `ExportImagesWithZones.wsdl` to locate and register the corresponding service, called “Service1”. For specific instructions on adding a web service, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

After registering the web service for this sample, add a new Web Services Hosting. When registering the new hosting for this sample, use port `4877` and locate this service at `http://localhost:4877/Service1.asmx`. For specific instructions on adding a new hosting, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

After adding a new Web Services Hosting, install the sample process using Intelligent Capture Administrator. For more information on the steps necessary to install a process, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

To setup the module steps in the WS Output sample:

1. Open the step settings for the WS Output step.
2. On the **WS Output** pane, specify the location of `ExportImagesWithZones.wsdl`.
3. Click **Parse** to populate the **Service Name** menu with a list of services.
4. Click **Mapping...** to open the virtual mapping window for the WS Output module. In the **Mapping** window, map some IA Value to the *`Operator`* parameter of the `ExportImagesWithZones` method (for example, map the *<Node_1>* IA Value *`SSOcr.Operator`* to *`Operator`*.)
5. Click **OK** to save these changes.
6. Select the **Scripting** pane to begin configuring the scripting settings of the WS Output module.
7. Configure the scripting page to use the script sample file named `WSOutput.ScriptSample.dll`

7.5 Reference

The topics within this section contain reference information useful while using the application in setup or production.

7.5.1 Windows

Windows topics in this section describe the user interface for working with client-side scripting.

7.5.1.1 Add Scripting File

Use the **Add Scripting File** window to import a script file and associate it with an Intelligent Capture module. From the module navigation pane, select **Scripting**, then click **Manage script files** to display the **Manage Scripts** window. Click **Add Script** to display the **Add Scripting File** window and add a compiled script assembly or data file.

Table 7-1: Add Scripting File Window

Element	Description
File to add	Select the script file to be added to the module step. Click **Browse** to locate the file or type the required path.
Script name	Automatically populated with the name of the selected file. If required, type a different name in the field.

Element	Description
This is a binary script file	Selected by default if the imported file is a .NET assembly file. For other file types, the default state is cleared. If the imported file is a data file to be used by the script, such as an *XML* file, clear this checkbox.
Automatically add other files for this assembly	Seeks additional files that are part of a multiple file assembly, then adds those files automatically. Although it is possible to create assemblies that span multiple files, it is unusual. Enable this checkbox to add multiple file assemblies if they are present. If no additional files are located, none will be added.

7.5.1.2 Manage Scripts

Display the **Manage Scripts** window by clicking the **Manage script files** button on the **Scripting Settings** pane. Use this window to view associated scripts, import a script, create a script, edit script properties, and delete unnecessary scripts.

Table 7-2: Manage Scripts Window

Element	Description
Scripts list	Displays information about scripts associated with the module step, such as the script **Name**, **Type**, **FileVersion**, **Assembly Version**, and **File Size**.
Create text script button	Displays the **Text Scripting File** window for writing scripts in C# or VB. NET (Visual Basic). This native script editor supports loading and editing an existing text script, creating a custom script from scratch, associating reference files with a script, and final compilation of the script.
Add file button	Displays the **Add Scripting File** window for importing a precompiled script *DLL* or a script data file. It is recommended that scripts be limited to 32767 characters.
Edit button	If an assembly file is selected, this button displays the **Script File Properties** window and properties of the selected script for modifying the script name or file type. If a text script file is selected, the script is opened in the **Text Scripting File** window for viewing, editing, or compiling.

Element	Description
Delete button	Deletes the selected script. A confirmation window displays. Click **Yes** to delete the script, or **No** to return to the **Manage Scripts** window. **Caution** Deleting a script can cause errors in batches that have already been created but have not yet been processed by the module. The error will occur if the batch settings contain a reference to a script file that has been deleted or renamed, because the module will not be able to find the required script when the batch is processed.
OK button	Saves changes in the **Manage Scripts** window and returns to the **Scripting Settings** pane.
Cancel button	Closes the **Manage Scripts** window without saving any changes and displays the **Scripting Settings** pane.

7.5.1.3 References

External assemblies used in a script must be associated with the script in Intelligent Capture. These references are stored in a separate file. In Visual Studio they are part of the project settings; in the Intelligent Capture script editor they are associated in the **References** window, displayed by clicking the **References** button in the "Text Scripting File" on page 475 window. Assembly references selected in this window can be saved as an *XML* document, or references contained in an external *XML* document can be loaded and saved. The code and the references together make up the script.

Table 7-3: References Window

Element	Description
References table	Displays information such as **Component Name**, **Version**, **Runtime** version, and assembly **Path**. This lists any assemblies referenced by the script and all assemblies referenced from the `HKLM\SOFTWARE\Microsoft\.NETFramework\AssemblyFolders`, `HKLM\SOFTWARE\Microsoft\.NETFramework\v2.0.50727\AssemblyFoldersEx` and `%WINDIR%\Microsoft.NET\Framework\v2.0.50727` registry keys. Assemblies can be added manually using the **Browse** or **Load** buttons. Any new assemblies will have the associated checkbox selected by default. To associate a listed assembly with the script, select the checkbox next to the name of the assembly.
Browse button	Select a *DLL* assembly file required by the script. The path to the selected file is displayed in the **Path** column of the table.
Load button	Select an *XML* file containing assembly information required by the script. The new assemblies are displayed in the list of components.
Save button	Saves the associated references to a file.
OK button	Saves changes and closes the **References** window.
Cancel button	Closes the **References** window without saving any changes.

7.5.1.4 Script File Properties

Displays basic properties of a script, including the script name and file type. Display the **Script File Properties** window by selecting a file from the script list in the "Manage Scripts" on page 469 window and clicking the **Edit** button.

Table 7-4: Script File Properties Window

Element	Description
Script file name	Displays the name of the selected file. To modify the file name, type a new name in the field. **Caution** Renaming a script can cause errors in batches that have already been created but have not yet been processed by the module. The error will occur if the batch settings contain a reference to a script file that has been renamed, because the module will not be able to find the required script when the batch is processed.
File type	Identifies the file type of the selected file: • **Binary assembly file**: Selected if the script is a .NET assembly *DLL*. If the file is not a *DLL* file, this option is unavailable. • **Data file**: Always available. Select this option if the file is anything other than a .NET assembly *DLL*, such as a data file used by the script. Many types of files can be used as a data files, including TXT, *XML*, or *DLL* files.
OK button	Saves changes and closes the **Script File Properties** window.
Cancel button	Closes the **Script File Properties** window without saving any changes.

7.5.1.5 Script Filter

Allows scripts to be enabled or disabled for use by the module, and automapping information contained in the script to be applied or disabled for the process or batch. Display the **Script Filter** window by clicking **Filter** on the **Scripting Settings** pane.

Table 7-5: Script Filter Window

Element	Description
Script Filter list	Displays all the script assemblies associated with the module step. To make a script available to the module, select the associated checkbox. Clear the checkbox to disable the script for this module step.
Apply auto mapping information	Applies automapping information defined in the selected scripts. Metadata can be included in a script allowing event mappings to be configured automatically, but automapping information in a script does not have to be applied to any particular process or batch. Clearing the checkbox instructs the module to ignore automapping information included in the script. If automapping is enabled, it may or may not be possible to map additional events manually, as this depends on how automapping has been applied in the script.
Select All button	Selects the checkboxes for all available scripts, enabling them for use by the module. To enable or disable particular scripts, select or clear the associated checkbox.
Clear All button	Clears the checkboxes for all available scripts, disabling them for use by the module. To enable or disable particular scripts, select or clear the associated checkbox.
OK button	Saves changes and closes the **Script Filter** window.
Cancel button	Closes the **Script Filter** window without saving any changes.

7.5.1.6 Scripting Settings

Select **Scripting** from the navigation panel to display the **Scripting Settings** pane and implement scripting features. Review existing event mappings, apply filters, access script file management features, and customize event mappings using the options on this pane.

Table 7-6: Scripting Settings Pane

Element	Description
Module event mappings	Associate module events with particular script actions. Mappings specified here apply to all processes and batches processed by the module on the Intelligent Capture Server to which it is connected. Map each event by selecting an available option from the **Event** list box, and pairing it with an action from the **Scripts** list box.
Task event mappings	Associate task events for a process or a batch. Task events are related directly to a task or step. If configured within a process these mappings apply to all new batches based on that process. Batches based on previous process settings are not affected. If configured for a batch, the mappings apply to that batch only. Map each event by selecting an available option from the **Event** list box, and pairing it with an action from the **Script** list box.
Customize button	Displays an editor for modifying custom parameters implemented by a script. Intelligent Capture provides a **Default Parameter Editor** and the ability to create a custom editor if needed. Refer to the *Client Scripting Guide* for help working with custom parameters.
Filter button	Displays the **Script Filter** window for enabling or disabling the action of individual scripts, and applying automapping information.
Manage script files button	Displays the **Manage Scripts** window for importing script files, creating scripts with the text editor, editing script properties, or deleting scripts from the module step.

7.5.1.7 Text Scripting File

The **Text Scripting file** window displays when you click **Create text script** from the **Manage Scripts** window.

Table 7-7: Text Scripting file Window

Element	Description
Script name	Type a name for the script. If an existing code file is loaded, this field is automatically populated with the script file name. **Caution** Renaming a script can cause errors in batches that have already been created but have not yet been processed by the module. The error will occur if the batch settings contain a reference to a script file that has been renamed, because the module will not be able to find the required script when the batch is processed.
Language	Select a .NET programming language for the script from the list box, either **C#** or **VB. NET**.
Script	Type the script in the **Script** box. Alternatively, click **Load** to open an existing script for editing, compiling, and saving.
References button	Displays the **References** window for creating references to external assemblies, saving your references to an *XML* file, and associating an existing *XML* file containing assembly references with the script.
Load button	Loads a code file to display in the **Text Scripting File** window for editing, compiling, and saving. Specify **C#** or **VB. NET** in the **Language** list box to locate *CS* or Visual Basic code files.
Save button	Displays the **Save As** window for specifying where the script should be saved. Typing a name in the **File name** list box automatically populates the **Script name** field in the **Text Scripting File** window.

Element	Description
Compile button	Compiles the .NET code, automatically creating the final assembly. If the script compiles without errors, a success message displays. If errors occur, the **Compile Errors** window displays a list of errors, including the error number, description and position in the script. Click **OK** to return to the **Text Scripting File** window.
OK button	If the script is unchanged, and has been previously compiled, this button closes the **Text Scripting File** window. If the script has changed since it was last compiled, this button automatically compiles the script and displays either a success message, or the **Compile Errors** window. If the compile was successful, click **OK** to return to the **Manage Scripts** window. Otherwise, fix the errors and try recompiling.
Cancel button	Closes the **Text Scripting File** window without saving any changes and displays the **Manage Scripts** window.

Chapter 8

Process Developer

8.1 Overview

This section presents users with conceptual information about Process Developer, as well as step-by-step instructions on how to use the application. The topics within this section cover basic overview information and introduce the features and functions of the application.

The specific capabilities of each client module are described in *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

8.2 Understanding Process Developer

Process Developer is an Intelligent Capture module that enables you to create processes using Visual Basic for Applications (VBA). An Intelligent Capture process is a detailed set of instructions. It directs the Intelligent Capture Server to route images and data to the appropriate client modules in a specific order. VBA is a programming language that is nearly identical to Microsoft Visual Basic, except that it cannot be used to create standalone applications. Process Developer closely resembles the VBA programming environment, but has been modified to work specifically with Intelligent Capture. Use Process Developer to develop, compile, install, and debug Intelligent Capture processes (*IPP*) in a programming environment.

Notes

- Because it is based on the Visual Basic for Applications SDK, Process Developer does not support mixed-mode regional settings. Configure the regional settings so that formatting and display language settings reference the same code page as the code page specified by the **Language for non-Unicode Programs** setting.
- If you have previously worked with *PCF* (*SBL*-based scripts used with previous versions of Intelligent Capture), you can use Process Developer to convert them to *IPP*.

Intelligent Capture also offers CaptureFlow Designer, which is a visual-based development environment. It is designed for administrators who are already familiar with the Intelligent Capture system and are responsible for creating and designing new Intelligent Capture processes.

As processes are developed in CaptureFlow Designer, the steps, default path, and conditional branches are displayed graphically. Design elements can be moved, copied, and pasted as needed. When processes are saved, CaptureFlow Designer generates an XPP process file that contains all of the information related to the process definition. CaptureFlow Designer also generates an *IPP* file that can be viewed in Process Developer and an *IAP* file that can be configured in Intelligent Capture Administrator. For more information on creating processes, see *OpenText Intelligent Capture - Designer Guide (ECPCORE-CPD)*.

Caution

- You cannot use Process Developer to debug batches that were created from processes that were developed (or upgraded) in CaptureFlow Designer 7.5.
- Modifications made to an *IPP* file using Process Developer cannot be viewed in CaptureFlow Designer.
- Modified *IPP* files using Process Developer must be maintained in Process Developer going forward. To avoid overwriting the *IPP* and *IAP* files that CaptureFlow Designer generates, save them to a different location.

8.3 Understanding MDFs and IPPs

To create a process, you create two types of Intelligent Capture files: *MDF*s and *IPP*.

An *MDF* defines the static IA Values that are associated with a particular module. Your project must include at least one *MDF* file for each module that the process uses. The process can read or write the IA Values defined in the *MDF*s.

When you add an *MDF* to a project, a copy of the *MDF* is written into the project file. After adding the *MDF* to the project, it is not necessary to have the original *MDF* available to continue working with the project.

The *IPP* is the VBA project file. It contains Visual Basic code from which the process is created. There is only one *IPP* file per project, and it contains all of the process code.

8.4 Language Considerations

Intelligent Capture supports data in multiple languages. To develop *IPP*s that can support multiple languages, use the following recommendations:

- Choose a single code page and use it as the system code page across all machines running Process Developer. If you choose not to heed this restriction, then use only *ASCII* characters for Visual Basic code and for naming processes, steps, IA Values, and variables. This means, for example, that you cannot use non-*ASCII* characters in direct literal assignments or in local variable names. Process Developer allows non-*ASCII* characters for these items. Intelligent Capture Server does not let you install a process with non-*ASCII* characters that was compiled on a machine having a different code page.

  Caution

  The Intelligent Capture Server cannot detect whether it is code page-compatible with pre-6.5 processes. Processes compiled on a pre-6.5 system with a different code page setting than the Intelligent Capture Server, can cause data corruption or server exceptions. Recompile and reinstall all processes that are being used in a mixed code page environment.

  These restrictions do not apply to department names or to the default values of IA Values defined in *MDF*s using *UTF-8* encoding. In other words, Unicode characters are supported in dynamic IA Value names. Also, none of these restrictions apply when the Process Developer machine and the Intelligent Capture Server are using the same code page.

- Use a *UTF-8* editor (for example, Windows Notepad) to define a custom data-only *MDF* to hold literal text values in multiple languages. *MDF*s can declare Unicode (*UTF-8*) values.
- In the custom data-only *MDF*, define variables for all literal text that use characters from languages that are not included in the specified system code page. The variable names themselves must use characters from the system code page only; however, the values can be in any language present in the system. For example, if the Process Developer system code page is 1252, name the variables with Latin alphabet characters. The values can mix alphabets and languages, such as Korean, Chinese, French, and Russian.
- Use only characters from the Process Developer system code page for the following:
  - Process names
  - IA Value names
  - Step names
  - Variable names

- You can define a process on a machine whose code page is different from the code page of the machine executing the process. In this case, do not include any literal strings with non-*ASCII* characters in the VBA code. If your environment has only a single code page, VBA literal strings can be defined without this restriction.

With these recommendations, you design *IPP*s that can run batches with any combination of codes pages and regional settings of Intelligent Capture Servers and client machines. Also read the upgrade considerations described in the *OpenText Intelligent Capture - Installation Guide (ECPCORE-IGD)*.

To ensure seamless multiple language/multiple code page compatibility, use CaptureFlow Designer instead of Process Developer to create your *IPP*s.

8.5 Differences between CaptureFlow Designer and Process Developer

The process development model for Intelligent Capture now includes CaptureFlow Designer, which enables you to create processes in a graphics-based environment. The following list outlines some of the key differences in functionality between CaptureFlow Designer and Process Developer:

- **Batch creation**: In CaptureFlow Designer, you can use multiple batch creating modules but only one of each type. This feature avoids an ambiguous batch-creation flow. Process Developer allows multiple instances of a batch creating module. However, only the first instance of the batch creating step is used when the batch is created; subsequent instances can never be triggered.
- **Process files**: CaptureFlow Designer stores processes as `XPP` files rather than as Process Developer *IPP* files.
- **MDF files**:
  - In CaptureFlow Designer, you add custom IA Values to a process in the **Custom Value** panel rather than to an *MDF* file. However, custom values are restricted to standard value types only, and cannot use user-defined object types. Additionally, you can reuse an existing data-only *MDF* from a Process Developer *IPP* instead of recreating custom values in CFD. A data-only *MDF* defines a custom module which is used only to store data when there is no actual module executable to run. for more information on how to add a custom module to a process, see *OpenText Intelligent Capture - Designer Guide (ECPCORE-CPD)*.
  - In Process Developer, you can use *MDF* files that are part of the *IPP* file or the *MDF* that resides on the Intelligent Capture system. CaptureFlow Designer does not provide the option to continue using the *MDF* files that are saved inside the process. CaptureFlow Designer automatically uses all of the Intelligent Capture client module *MDF*s that are installed on the CaptureFlow Designer machine, showing their modules in the **Steps** panel.

When you open an XPP file in CaptureFlow Designer, the *MDF* files that are located on the CaptureFlow Designer machine are always used. If the *MDF* files on the hard drive are updated, the *MDF* files in CaptureFlow Designer processes are updated when they are opened. In this event, a message displays and you can close the file without saving it to avoid an automatic update.

When you add a module from the **Steps** panel to the process flow, CaptureFlow Designer adds the associated *MDF*s to the XPP in a modified form. When CaptureFlow Designer generates the *IPP* and *IAP* files, it creates new *MDF* files. The new *MDF* files contain different trigger information and other supporting values needed for the generation of the XPP file. When CaptureFlow Designer generates the *IPP* and *IAP* files, new *MDF*s are created. They contain trigger IA Values based on how work progresses in the process flow in the design pane. Existing *MDF* triggers, including the <*Ready*> IA Value, are converted to standard non-trigger <*Input*> IA Values.

- **Task Execution**: CaptureFlow Designer does not support parallel execution of tasks for a given batch node. The tasks are serialized. By choosing a task level lower than 7, you can perform parallel execution among different nodes.
- **Task Routing**:
  - Unlike Process Developer, CaptureFlow Designer creates processes that perform tasks exactly as designed in the Design pane.
  - CaptureFlow Designer uses decision blocks and jumps for alternate flows, and IA Values to determine which flow to take.
- **Value Assignments**: In CaptureFlow Designer, value assignments have intelligent iterations and do not require additional “For loop” logic to assign values at a lower level.
- **Conditions and Expressions**: CaptureFlow Designer uses a new generic set of operators and functions for conditions and expressions that are different than the VBA equivalents used in Process Developer.

8.6 Designing

Topics in this section help administrators customize the application to meet business requirements.

8.6.1 Working with MDFs

This section describes *MDF*s and explains how to use them in *IPP*.

8.6.1.1 Viewing an Existing MDF

The *MDF*s for modules that are included with Intelligent Capture are located in the `Client\src\ipp\` subfolder of the Intelligent Capture installation folder (for example, `C:\Program Files\InputAccel\Client\src\ipp\`). An *MDF* is a plain text file that you can open with any text editor, such as Windows Notepad. You can view the code for these *MDF*s to understand how to create your own.

8.6.1.2 Creating an MDF

If the *MDF* provided with Intelligent Capture does not meet your needs, create an *MDF* for a custom module or an existing Intelligent Capture module.

To create an *MDF* for a custom module:

1. Use any text editor, such as Windows Notepad, to create a text file.
2. Declare the modules and IA Values needed for your process.
3. Save the file with the extension `.mdf`. Process Developer can only open *MDF*s with this extension.

   Make sure that your text editor does not add another extension to the file. Windows Notepad, for example, adds a text file extension, creating a file ending in `.txt`. To avoid an extension being added, enclose the file name in quotation marks when you save it. Or, rename the file after it has been created.

Caution

Limit *MDF* names to 27 characters, including the extension. Process Developer accepts longer names, but longer names can cause problems.

8.6.1.3 Creating an MDF to Store Common Values

If you want to add IA Values to your project that are not related to a particular module, then you can create a custom module. This custom module does not correspond to an actual program that runs on a computer. Add a step of this custom module in your *IPP* to store common values. Common values are accessible in Process Developer while developing your *IPP*. These values are also available to anyone setting up the steps in your process.

To create a custom module, simply create an *MDF*, following the instructions in "Creating an MDF" on page 482. Select a unique module name. Do not use a same name to include several modules in your project. For the maximum length allowed, refer to the *System Limits* topic in the *Supplemental Reference*.

8.6.1.4 Configuring the Include Directory for MDFs

Process Developer uses the **Include** environment variable to specify in which directories to look for *MDF*s. Separate multiple directory paths with semicolons. For example, setting **Include** to `\\iaclient\src\ipp;c:\myMDFs` specifies the directory `\\iaclient\src\ipp` and the directory `c:\myMDFs`.) If another value exists for **Include**, then it is appended with the default value.

If Process Developer does not display the *MDF*s that you have installed on your system in the **Select MDFs** window, reset the **Include** variable.

To reset the Include variable:

1. On the workstation where Process Developer is installed, click **Start**, select **Settings** > **Control Panel**. The **Control Panel** window displays.
2. Double-click the **System** icon. The **System Properties** window displays.
3. Select the **Advanced** tab, and then click **Environment Variables**. Two lists of environmental variables are displayed: **System variables** and **User variables**.
4. From the **User variables** field, select the variable **Include**, and then click **Edit**. (Or, click **New** to create a variable.)
5. In the **Variable value** field, type the paths and click **OK**. These paths are where your *MDF*s are installed.
6. Restart **Process Developer** for the settings to take effect.

Related Topics

"Viewing an Existing MDF" on page 482

8.6.1.5 Modifying an Installed MDF

Caution

Before you modify an *MDF*, make a copy in a different location. Edit the copied *MDF* so that you do not lose your changes when a new version of Intelligent Capture is installed.

If you want to add new values for a specific module, create an additional *MDF* for the module. Then, declare the new values in this second file. Declarations for the same module in multiple *MDF*s are merged when you import them into an *IPP*, effectively turning them into a single module declaration.

You cannot declare duplicate IA Values at the same level for the same module. Be careful not to redeclare any values already declared in the original *MDF*.

8.6.1.6 Updating an IPP with a Changed MDF

If you modify an *MDF* included in an *IPP* without updating this *MDF* in your project, Process Developer displays an error message.

To continue using the original *MDF*, select **Use Project MDFs**. Process Developer opens the *IPP* using the *MDF* that is stored in the project.

To update an *IPP* to use the new, updated version of the *MDF*:

1. Select **Select MDFs**. The **Select MDFs** window displays with both versions of the *MDF* displayed:
   - Original *MDF*: The original version of the *MDF* is displayed at the top of the **Modules List**.
   - Updated *MDF*: The newer version of the *MDF* is displayed at the bottom of the **Modules List**. It is initially selected and its path is displayed in the **Directory** column.
2. To use the newer version of the *MDF*, click **OK**. The **Modify Steps** window displays.
3. Edit the project steps, if necessary.
4. Click **OK**.

8.6.1.7 Extracting an MDF from an IPP

You can edit an *MDF* included in an *IPP* even if you do no longer have the original file *MDF*. Extract the *MDF* code out of the process into a new *MDF* file and edit it from there.

To extract an *MDF* from an *IPP*:

1. Open the process file in Process Developer.
2. From the **File** menu, select **Modify Steps**.
3. In the **Modify Steps** window, click **Select MDFs**.
4. The *MDF*s that are used in the *IPP* show **Project** in the **Directory** column. Click this column header to sort the list by directory location.
5. Select the *MDF* you want to extract, and then click **View**.
6. In the **View MDF** window, select all of the text, and then type **CTRL + C** to copy the selected text to the Windows clipboard.
7. Using any text editor, open a new text file, and then paste the clipboard contents into the empty file.
8. Save the file with the extension `.mdf`.

8.6.1.8 Creating MDFs for FormWare Jobs

If you have FormWare for InputAccel installed on the same machine as Process Developer, additional FormWare-specific commands appear on Process Developer's **Tools** menu enabling you to create Intelligent Capture processes to interact with FormWare jobs. To display the FormWare menu commands, first create *MDF*s for your FormWare jobs.

To create *MDF*s for FormWare jobs:

- Run Process Developer, and then select one of the following options:
  - **Update MDF for All Jobs**: Causes Process Developer to generate *MDF*s for every FormWare job in the `FormWare Jobs` folder. If an *MDF* exists for a particular job and the job file is modified since the *MDF* was last saved, then Process Developer updates the *MDF*.
  - **Create MDF for a Job**: Causes Process Developer to create an *MDF* for one job. Select the *JDF* file from the list of jobs located in the `\FormWare\Jobs` folder that displays when you select this option.

FormWare *MDF*s define the following global IA Values:

- *<FWJobFile>* contains the name of the job file that is currently associated with the batch. Every FormWare for InputAccel module uses this value when creating *BDF*s for processing.
- *<IABatchComboChoices>*, *<IABatchComboTitle>*, and *<IABatchComboResult>* are used to present a list of available FormWare jobs for setting up an Intelligent Capture batch.

The *FormWare File System Installation Guide* and *FormWare for InputAccel Guide* contain additional information.

8.6.1.9 Understanding MDF Programming Concepts

This section describes the general structure and syntax of an *MDF*.

8.6.1.9.1 Understanding the Structure of an MDF

*MDF*s can contain only declarations. Code that defines process logic and behavior must be placed in *IPP*.

An *MDF* is divided into the following sections:

- **Object and IAVariant declarations**: You can declare one or more objects and/or IAVariants. Objects and IAVariants are composite data types that, if used, must be defined outside of any module declarations.
- **Globals value declarations**: You can optionally declare a global values section. The Globals section defines IA Values that are not associated with a particular module. In a batch, these values appear as non-nodal values. A global IA Value

must be of type `String`, and it can have optional initial value data. Here is an example of how to declare global values in an *MDF*:

```
Globals
    Value1 as String = "1234"
    Value2 as String = "XYZ"
    Value3 as String
End Globals
```

The full context of a global value in an *IPP* is `pRoot.Tree.Globals.GlobalValue`. You can declare the same global value twice, but only if the declarations are identical. Declaring the same value twice is useful to reference, within a single *IPP*, multiple *MDF*s with overlapping global declarations. The system predefines several global value names that can be used in any *IPP*. The "IA Values" on page 570 section contains information about predefined global IA Values.

- **Module declarations**: You can declare one or more modules in a single *MDF* and set specific node levels to which the module IA Values apply.

  The syntax of a module declaration is as follows:

  ```
  Module <ModuleID>[<(Level>)]
      IA Value Definitions
  End Module
  ```

  where:

  – *<ModuleID>* is the code identifier for the module. It is the name of the module executable file without the filename extension.

  – *<Level>* indicates the level at which the IA Values in this section are defined. It can be either a single integer from `0` to `7` inclusive, or the letter *<T>*. *<T>* indicates the trigger level specified when the module is added to an *IPP*. If *<Level>* is omitted, the values are assigned to level 0.

Here is an example of a simple *MDF* with multiple module declarations:

```
Module SCANPLUS(0)
    OutputImage as     File,Output
End Module

Module SCANPLUS(0)
    Level_0_KeyEntry_0 as String,Output
End Module

Module RSCANPLS(0)
    InputImage as File,Input
    OutputImage as File,Output
    RescanReason as String,Input
End Module
```

8.6.1.9.2 Declaring IA Values

IA Value declarations in an *MDF* use the following syntax:

```
ValueName as DataType[<,Attribute1> [,<Attribute2>[<...>]]][=InitialData]
```

The optional attributes determine how the value gets used. The "MDF Value Attributes" on page 549 section contains a list of IA Value attributes and their meanings.

Consider the following when declaring IA Values in an *MDF*:

- Each IA Value must be declared on a single line.
- Each IA Value must be declared on its own line.
- *MDF*s do not use any statement delimiters other than line breaks.
- White-space characters (sequential spaces, tabs, and extra line breaks) are automatically consolidated.
- As with Visual Basic, *MDF* syntax is not case sensitive, so it does not matter whether you use uppercase or lowercase letters. However, it is good practice to be consistent when naming and referencing values.
- *MDF*s can contain only declarations. When you import an *MDF*, any initial data declared for your values are assigned in the `Process_PreInstall` procedure of the *IPP*. Instead of declaring static initial data, you can assign that data programmatically. For example, you can have data from one value based on another value. First, declare the value in the *MDF*. Then, create code in the *IPP* to assign the data. Use the following *IPP* procedures to initialize data in your code:
  - `Process_Install`
  - *<Step>*`_Prepare` (where *<Step>* is the name of a step object)

8.6.1.9.3 Using MDF Values

This section describes how to assign data types, attributes, and levels to *MDF* Values.

Integer

An Integer is a 16-bit integer with a valid range of -32768 to 32767. Attempting to set an Integer *MDF* value to a value outside of this range causes a runtime error. Attempting to use an IAValueSet function to set anInteger outside of this range truncates the 32-bit number to 16-bits. For example, a number such as 33000 is truncated to -32536.

The following code demonstrates how to use Integer *MDF* values using the Process Developer Object Model (intellisense):

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim iTemp as Integer
    p.Step.MyInteger = 5000
```

```
    iTemp = p.Step.MyInteger + 500
End Sub
```

The following code demonstrates how to use Integer *MDF* values using `IAValueSetLong` and `IAValueGetLong` methods:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim iTemp as Integer
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" &
p.Tree.NodeID & "/$step=Step/"
    Call IAValueSetLong (strKey & "MyInteger", 14000)
    iTemp = IAValueGetLong (strKey & "MyInteger", 0)
End Sub
```

The following code demonstrates invalid ways of setting Integer *MDF* values:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$step=Step/"

    p.Step.MyInteger = 150000 'This  cause a run-time error

    Call IAValueSetLong(strKey & "MyInteger", 150000 ) 'This sets MyInteger to 18928
(the lower 16 bits of the Long value 150000)

End Sub
```

Long

A Long is a 32-bit integer with a valid range of -2147483648 to 2147483647.

The following code demonstrates valid ways of using Long *MDF* values:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim lTemp as Long
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$step=Step/"

    p.Step.MyLong = 145000

    Call IAValueSetLong(strKey & "MyLong", 14000 )
    lTemp = p.Step.MyLong -100000
    lTemp = IAValueGetLong( strKey & "MyLong", 0 )
End Sub
```

Boolean

A Boolean value can be set to false, true, 0 (meaning false), or 1 or -1 (meaning true). Boolean values are accessed with the `IAValueGetLong` and `IAValueSetLong` methods.

Note: Be especially careful when using `IAValueSetAscii` to set Boolean variables. Only a value of 0 is considered false. All other values are considered true. If you try to use `IAValueSetAscii` with a VB Boolean argument, then the Boolean is converted to a string (for example, “True” or “False”), and the Boolean *MDF* value is always set to true.

The following code demonstrates valid ways of using Boolean *MDF* values:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim bTemp as Boolean
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$step=Step/"
    p.Step.BValue0 = -1  'sets the boolean value to true
    p.Step.BValue1 = false
    Call IAValueSetLong( strKey & "BValue1", 1)  'sets BValue1 to true
    bTemp = IAValueGetLong( strKey & "BValue1", 0)
End Sub
```

File

A <*File*> is a reference to a stage file on the Intelligent Capture Server. In *IPP*, files are typically used as a means of triggering modules. Triggering is made by setting an input file of one module to the output file of another module.

For files within an *MDF*, input files are automatically considered triggers, unless the `Notrigger` keyword is specified.

For files within an *IPP*, setting a <*File*> IA Value to another <*File*> does not create a copy of that file. It simply sets the stage number of the <*File*>, so that the module finds the correct stage file. Output <*File*> *MDF* values get a default value when the *IPP* is compiled, to ensure that each output <*File*> has a unique value. Changing the value of an output <*File*> to another stage number can make that output file point to an output file of another module. If this change is made before the module runs, then the output file can overwrite the output file for another module.

A <*File*> IA Value must only be set to another <*File*> IA Value on the same node. The value of the <*File*> is an integer, which represents the stage number within the node to use. Because it is an integer, it cannot be set to a stage file from a different node. Attempting to do so actually sets the <*File*> to that stage number on the current node.

String

Within an *IPP*, a <*String*> *MDF* value functions much like a VB String. However, be careful with strings that contain special characters, especially the null character, `Chr$(0)`. Some modules are unable to read the portion of the string past the special character that marks an end of a string in the C language.

The following code demonstrates valid ways of using <*String*> *MDF* values:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim strTemp as String
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$step=Step/"

    p.Step.MyString = "Hello"
    Call IAValueSetAscii( strKey & "MyString", "New Value" )
    strTemp = p.Step.MyString & " with appended text."
    strTemp = IAValueGetAscii( strKey & "MyString", "no value" )
End Sub
```

String*n

Within an *IPP*, a <*String**><*n*> is a fixed-length string, where <*n*> represents the number of characters in the string. Generally, it is more efficient to use a fixed length for strings of less than 15 to 20 bytes.

A fixed-length string can be defined in an *MDF* with the optional attribute, *`Format`*. The Intelligent Capture Server formats the <*String*> data at runtime according to this *`Format`* attribute. The *`Format`* attribute can be set to a number in the range `0` to `7`. If *`Format`* is not indicated in the *MDF*, then it is set to the default `0`.

The following table is a list of *`Format`* settings:

Table 8-1: String*n *`Format`* Settings

Value	Description
0	Apply left justification, " " fill, " " fill when no data
1	Apply Right justification, " " fill, " " fill when no data
2	Apply Left justification, "0" fill, " " fill when no data
3	Apply Right justification, "0" fill, " " fill when no data
4	Apply Left justification, " " fill, "0" fill when no data
5	Apply Right justification, " " fill, "0" fill when no data
6	Apply Left justification, "0" fill, "0" fill when no data
7	Apply Right justification, "0" fill, "0" fill when no data

In the previous table, a combination of three properties is available for *`Format`* settings:

1. **Justification (right or left)**: This property describes whether large values are truncated on the left or the right when assigned to this fixed-length string. It also describes whether fractional numeric values are truncated on the left or right.
2. **Fill Character with data present**: This property describes whether to use a space or a zero character to pad values. Padding is for values that are smaller than the IA Value length, when the value being assigned is not empty.
3. **Fill Character with no data present**: This property describes whether to use a space or a zero character to fill the IA Value when the value being assigned is an empty string.

Note: Most modules do not validate fixed string lengths. Loss of data can occur.

Single

A <*Single*> is a 32-bit floating point real number with approximately seven significant digits of precision. Because the significant digits represent real numbers, access <*Single*> values with `IAValueGet/SetAscii`, not `IAValueGet/SetLong`.

The following code demonstrates valid ways of using <*Single*> *MDF* values:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim fTemp as Single
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$instance=Step/"

    p.Step.MySingle = 14.534
    Call IAValueSetAscii(strKey & "MySingle", "13.6478" )
    fTemp = p.Step.MySingle - 1.1
    fTemp = IAValueGetAscii(strKey & "MySingle", "0.0" )
End Sub
```

Double

A <*Double*> is a 64-bit floating point real number with approximately 15 significant digits of precision. Because the significant digits represent real numbers, access <*Double*> values with `IAValueGet/SetAscii`, not `IAValueGet/SetLong`.

The following code demonstrates valid ways of using <*Double*> *MDF* values:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim fTemp as Double
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$step=Step/"
    ' MySingle  be rounded to 14.0, because Singles only
    ' have about 7 digits of precision. MyDouble will be set
    ' correctly.
    p.Step.MySingle = 14.0000000001
    p.Step.MyDouble = 14.0000000001
    Call IAValueSetAscii(strKey & "MyDouble", "11.3438" )
    fTemp = p.Step.MyDouble - 2.100487
    fTemp = IAValueGetAscii(strKey & "MyDouble", "0.0" )
End Sub
```

Note: Setting an IA Value of type <*Double*> close to its minimum or maximum limit can cause an error.

Currency

<*Currency*> is an 8-byte scaled integer. Because the integer represents fractional currency amounts, access <*Currency*> IA Values with `IAValueGet/SetAscii`. If you intend to use only whole currency units, then `IAValueGet/SetLong` can be used as well.

The following code demonstrates valid ways of using <*Currency*> *MDF* values:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim CTemp as Currency
    Dim strKey As String
```

```
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & _ p.Tree.NodeID & "/$step=Step/"

     p.Step.MyCurrency = 10.45
    p.Step.MyCurrency = "$20.15"

    ' MyCurrency will be rounded to $23.03
    p.Step.MyCurrency = FormatCurrency(23.034)
     ' MyCurrency will not be rounded. The second parameter
    ' of FormatCurrency sets the number of digits after the
    ' decimal point.
    p.Step.MyCurrency = FormatCurrency(23.034, 3)
    Call IAValueSetAscii(strKey & "MyCurrency", "$11.34" )
    Call IAValueSetLong(strKey & "MyCurrency", 11 )
    CTemp = p.Step.MyCurrency
    CTemp = IAValueGetAscii(strKey & "MyCurrency", "$0.00" )
End Sub
```

Date

A <*Date*> is an 8-byte date, like a VB date. Dates can be set to a date only, a time only, or a date and a time. Within an *IPP*, dates are set by enclosing a date or time within # symbols. Because dates are stored internally in a floating point format, access <*Date*> IA Values with `IAValueGet/SetAscii`.

The following code demonstrates valid ways of using <*Date*> *MDF* values:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim DTemp as Date
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$step=Step/"

    p.Step.MyDate = #July 4, 1976#
    Call IAValueSetAscii(strKey & "MyDate", "10:46 PM")
    p.Custom.MyDate = Now
    DTemp = p.Custom.MyDate
    DTemp = IAValueGetAscii(strKey & "MyDate", "Jan 1, 1970 12:00:01 AM") End Sub
```

Object

An Object is like a C structure or VB type. Objects contain a set of member IA Values that are accessible by name. These members can be any IA Value type except <*File*>; members can even be another <*Date*> or IAVariant. For example, note the use of the "Address" Object in `HouseObject` in the following code sample:

```
Object AddressObject
    Number as Long
    Street as String
    City as String
    ZIP as Long
End Object

Object HouseObject
    Address as AddressObject
    Owner as String
    SqFt as Single
End Object

Module Custom
    Residence as HouseObject
    Comments as String
    Price as Currency
End Module
```

All values start with a default value. For strings, the default value is the empty string “”. For fixed-length strings, the default value is the value as if set to “”. Depending on the formatting, this value can be padded with 0 or “”. For numeric variables, it is 0.

Objects do not take up much space (4 bytes or so) until first accessed. So, if an object is declared within an *MDF* and never used, then it does not make the batch larger.

Accessing an Object through IAValueGet/Set Functions

An Object cannot be directly accessed through `IAValueGet/Set` methods, but its member variables can be accessed. Member variables are separated from their parent Object by a colon (:), not a period (.). For example:

```
CallIAValueSetAscii("$batch=" & p.Tree.BatchID
& "/$node=" & p.Tree.NodeID & "Step.Residence:Address:Street","Parkmoor")
```

Caution

If you use the `StepName.Value` syntax, then only the portion of the string after `StepName` is considered the key. If you are using an Object or IAVariant, then the parent object, the child IA Value, and the ':' (colon) separator are also included in the key.

Rules for Object Behavior within an *MDF*

- Circular references are not allowed. That is, if an object contains another object, then it cannot contain the parent object and none of the child objects can contain the parent object. For example, if `ObjectA` has a member of type `ObjectB`, then `ObjectB` cannot have a member of type `ObjectA`.
- There is one reserved member, Type, which is a string that always displays in each object. Type is declared as a constant with a value of the Object name.
- Top-level objects can contain up to 16384 members. The highest level *MDF* Values of type Object can contain up to 16384 member variables. This value is an Object whose parent is not another Object *MDF* Value, but is a module step. For a top-level IAVariant, the same rule applies to the Object members of that IAVariant.
- A lower-level Object can contain up to 1000 members. Any *MDF* value of type Object whose parent is also an Object *MDF* Value, can contain up to 1000 member variables. For an IAVariant whose parent is an Object *MDF* Value, the Object members of that IAVariant can contain up to 1000 member variables.
- Objects or their children cannot be triggers.
- Object and IAVariant members can be nested up to five levels deep. Nesting means that an Object can contain an IAVariant member variable, which can in turn contain an Object member variable, and so on. The following example shows Objects nested five deep, where each `Obj` *<N>* is an Object:

```
p.Step.Obj5.Obj4.Obj3.Obj2.Obj1.IntValue = 25
```

Caution

If you use the `StepName.Value` syntax, then only the portion of the string after `StepName` is considered the key. If you are using an Object or IAVariant, then the parent object, the child IA Value, and the ':' (colon) separator are also included in the key.

- You can use a temporary *COM* object as a parameter type for a function or subroutine. Changes made in the function directly change the IA Values referenced. For example:

```
Private Sub ChangeHouse(House As IHouseObject)
    House.Address.City = "San Diego"
    House.Owner = "John Doe"
End Sub
```

```
Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    ' The following lines set:
    ' p.Step.Residence.Address.City to "San Diego"
    ' p.Step.Residence.Owner to "John Doe"
       Call ChangeHouse(p.Step.Residence)
End Sub
```

Rules for Object Behavior within an *IPP*

- Objects appear within an *IPP* as *COM* objects with all the member variable names appearing as properties.

```
p.Step.Residence.Address.Number = 1299
```

Note: The auto-fill feature displays all valid member IA Values.

- This *COM*Object type is an interface named `I<ObjectName>`. An Object can be stored in temporary variables. For example, given Step as a step of the Custom module defined in the sample *MDF*:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim obj as IAddressObject

    Set obj = p.Step.Residence.Address
    ' The next line changes p.Step.Residence.Address.Number
    ' to 1299.
    obj.Number = 1299
End Sub
```

- When using a temporary Object, note the following:
  - The temporary variables directly reference the original IA Values. Thus, changing a member variable of a temporary object changes the same member IA Value in the batch.
  - The temporary variables cannot be stored for longer than the event (`Prepare`, `Finish`, `Error`) that was invoked. That is, no global Object variables are allowed.
  - These objects can become invalid when an IAVariant changes type. The "IAVariant" on page 496 section contains more information.
- You can iterate through *Object* fields to remove the need for hard-coding names of fields in the *IPP*. The *MDF* type Object supports the `IIAValuesCollection`

interface, and this interface supports *OLE* Automatic Collection. The following required members are implemented:

- *Count*: Returns number of elements in the Values collection
- *_NewEnum*: Returns an iterator *COM* object that provides the interface `IIAValuesIterator` that has the following properties:
  - *Name*: String that is the name of the object member value.
  - *Type*: Read-only value that is a string giving the type of the value, like VB function `TypeName()`. It can be a Long, String, or an object name such as `FWDATA`.
  - *Value*: Read/write variant containing the current value.
- *Item*: Supports array-like access

The following example illustrates iterating through Object fields:

```
Dim Field As IIAValuesIterator
Dim Fields As IIAValuesCollection

Set Fields = p.Export.InputData.Form.Any

For Each Field in Fields
    If Field.Name = "AccountNumber" Then
        Field.Value.Data = "012-34-5678"
        Field.Value.Flagged = True
    End if
Next
Set Field = Fields (3)
```

The following are field iteration rules:

- Values are iterated in the same order as they are defined in the *MDF*.
- Array-like access uses *Index* as Long, starting from 1.
- Collection includes all defined values including const values.
- `IIAValuesCollection` of IAVariant includes all members defined in the *MDF* for this type, regardless of the current value. If it is not the current value for this IAVariant, `IIAValuesIterator` member `Value()` returns an empty Variant.

- Within an *IPP*, it is possible to assign an entire object to another, provided each Object is the same type. For example:

```
p.Step2.ObjA = p.Step1.ObjA
```

Using the `objects1.ipp` Code Sample

The sample *IPP* `objects1.ipp` demonstrates how to useObject and IAVariant data types. To run this sample, load it in Process Developer. The Main module includes a lengthy comment section with instructions on how to use the sample. After compiling to an *IAP* and installing the process on the Intelligent Capture Server, be sure to load the process settings from the file `objects1.txt`.

IAVariant

An IAVariant is like a VB variant or a C union. An IAVariant can have different data types depending on the data being stored. These data types must all be Objects. An IAVariant can hold data for several different Object types. As with Object, first define an IAVariant, and then declare a value that uses the new IAVariant as a data type. An IAVariant can have multiple declarations (the IAVariant is the union of the individual declarations). IAVariants are declared in the *MDF* as follows:

```
Object Employee
    Name as String, Output
    Age as Long, Output
    Salary as Currency, Output
    JobTitle as String, Output
End Object
```

```
Object Athlete
    Name as String, Output
    Age as Long, Output
    Sport as String, Output
    Rank as Long
End Object
```

```
IAVariant Person
    Employee
    Athlete
End IAVariant
```

```
Module Custom(n) 'n can be 0 to 7 (0 in the samples).
    MyFriend as Person
End Module
```

There must be at least one Object listed inside the `IAVariant` definition. These types must all be Objects declared elsewhere within the *MDF* (or within another *MDF* that is included in the project). An Object can have multiple declarations, but the declarations must be identical (including variable order). An `IAVariant` can hold only one of these values at any given time. The two reserved property names, <*Any*> and <*Type*>, cannot be used as Object types in the IAVariant definition.

Each of the Object types displays as a property within the IAVariant named the same as the type name. In addition, the two reserved properties, <*Type*> and <*Any*>, can be accessed both within VB and `IAValueGet/Set` methods. The property, <*Type*>, is a String used to select which of the types is active. The property, <*Any*>, is an object used to access the variables regardless of the current type. Within the *IPP*, the <*Any*> property is a COM object that supports late binding. It also contains as its members all variables that are common to all the objects within the variant.

Accessing an IAVariant through `IAValueGet/Set` Functions

Separate member types with a colon (:), not a period (.). For example:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim strTemp as String
    Dim lTemp as Long
    Dim strKey As String
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$step=Step/"
    Call IAValueSetAscii(strKey & "MyFriend:Employee:JobTitle", "Engineer")
    lTemp = IAValueGetLong(strKey & "MyFriend:Any:Age", 0)
End Sub
```

Rules for IAVariant Behavior within an *MDF*

- An IAVariant definition can only contain a list of Object types. No primitive types (such as, Integer, Long, String, and so on) or IAVariant types are allowed in this list.
- Objects and IAVariants can be nested up to five levels deep. Nesting means that an IAVariant can contain an Object member variable, which can in turn contain an IAVariant member variable, and so on. The following example shows IAVariants and Objects nested five deep, where each `Obj<N>` and IAVar<*N*> is an Object or IAVariant, respectively:

```
p.Step.IAVar5.Obj4.IAVar3.Obj2.Obj1.StrVal = "A"
```

Caution

If you use the `StepName.Value` syntax, then only the portion of the string after `StepName.` is considered the key. If you use Objects or IAVariants, then the parent object, the child IA Value, and the ':' (colon) separators are also included in the key.

Rules for IAVariant Behavior within an *IPP*

Rules for IAVariant behavior within an *IPP*:

- As with Objects, IAVariants appear within an *IPP* as *COM* objects with all the member variable names appearing as properties.
- This *COM* object type is an interface named `I<IAVariantName>`. IAVariants can be stored in temporary variables of this type. As with Objects, these temporary variables directly reference the original IA Value. Therefore, changing a member variable of a temporary IAVariant changes the same member IA Value in the batch.
- You can also use this temporary *COM* object type as a parameter type for a method or event handler. Changes made in the method or event handler directly change the IA Values referenced. For example, given <*Step*> as a step of the Custom module defined in the sample *MDF*:

```
Private Sub ChangePerson(Who As IPerson)
    Who.Any.Name = "John Smith"
End Sub
```

```
Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.Step.MyFriend.Type = "Athlete"
' The following line will set
'p.Step.MyFriend.Any.Name ' to "John Smith"
    Call ChangePerson(p.Step.MyFriend)
End Sub
```

- An exception to using `IObjectType` variables is that there is not an IA type for the <*Any*> property of IAVariants. Because of late binding of the <*Any*> property, it actually is of type `I<CurrentType>`, where *`CurrentType`* is the current type of the IAVariant. To store the <*Any*> property with a temporary variable, or to pass this property to an event, use a variable of the VB type Object. For example,

```
Private Sub ChangePerson(AnyPerson As Object)
    AnyPerson.Name = "John Smith"
End Sub
```

```
Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.Step.MyFriend.Type = "Athlete"
' The following line will set
' p.Step.MyFriend.Any.Name to "John Smith"
    Call ChangePerson(p.Step.MyFriend.Any)
End Sub
```

The previous examples represent the Custom module in the sample *MDF* and demonstrate the auto-fill feature for IAVariants. When using the previous example in Process Developer, the auto-fill feature is enabled while typing "`MyFriend`". A list of properties for MyFriend displays. When typing `Any`, the auto-fill feature is enabled and a list of the common properties between types are included in the IAVariant definition. "Salary" is not listed under the Any type because it is not a common value to all types. `Any.Salary` can be used in the *IPP*, but if the type of the MyFriend IAVariant is not Employee when the code is run, then a run-time error occurs.

Using the *CurrentType* of an IAVariant

- An IAVariant starts out in a non-typed state in which none of the types are active.
- Changing the type, *CurrentType*, discards the value of all the member variables already defined.
- Setting *CurrentType* to the same type does nothing.
- *CurrentType* can be set explicitly using the <*Type*> property. Setting the <*Type*> property to a value that is not a member of the IAVariant causes an error. Setting the <*Type*> property to an empty string sets the variant to a non-typed state.

Note: If the type changes within an *IPP*, then any subobjects (or sub-IAVariants) stored in VB variables become invalid and using them results in an error. This error is illustrated in the following example:

```
dim emp as IEmployee
set emp = p.Step.MyFriend.Employee
p.Step.MyFriend.Type = "Athlete"
emp.JobTitle = "Manager" ' should cause error
```

- A Get method (for example, `IAValueGetAscii`) cannot set *CurrentType* in any circumstance.
- A Set method (for example, `IAValueSetLong`) can set *CurrentType* in most circumstances.

Here are some examples of getting and setting IAVariant member values within an *IPP*, using the `IAValueGet/Set` methods. *<Step>* is a step of the Custom module defined in the sample *MDF*:

```
Private Sub Step_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim s as String
    Dim strKey As String
```

```
    strKey = "$batch=" & p.Tree.BatchID & "/$node=" & p.Tree.NodeID & "/$step=Step/"
    p.Step.MyFriend.Type = "Employee"

    ' The following line will succeed because of late binding.
    s = IAValueGetAscii(strKey & "MyFriend:Any:JobTitle", "Unemployed")

    ' The following line will fail because MyFriend is not of
    ' type "Athlete"; the default value should be returned.
    s = IAValueGetAscii(strKey & "MyFriend:Athlete:Sport", "Spectator")

    ' The following line will succeed because a "Set" can set the
    ' type of the IAVariant.
    Call IAValueSetLong(strKey & "MyFriend:Athlete:Rank", 27)
End Sub
```

Reading Values in an IAVariant

- Reading from the <*Any*> property uses the branch of the IAVariant selected by *`CurrentType`*. Trying to read a property that does not exist results in an error.
- Reading from a branch of an IAVariant that is not being used results in an error.
- Reading from an untyped IAVariant results in an error.
- Reading the <*Type*> property when in a non-typed state returns an empty string.

Here are some examples of reading IAVariant member values within an *IPP*. *`<Step>`* is a step of the Custom module as defined in the sample *MDF*.

```
Dim s as String
Dim temp as Long

' The following should return "", because MyFriend is untyped
s = p.Step.MyFriend.Type

' This should return an error because MyFriend is untyped.
s = p.Step.MyFriend.Any.Name
p.Step.MyFriend.Type = "Employee"

' The following line will succeed because of late binding.
s = p.Step.MyFriend.Any.JobTitle

' The following line will fail because "Sport" is a property
' of the "Athlete" type, and the current type is "Employee"
s = p.Step.MyFriend.Any.Sport

' This line  fail because the current type is Employee
temp = p.Step.MyFriend.Athlete.Rank
```

Writing Values to an IAVariant

- Setting a member variable of the <*Any*> property uses the branch of the IAVariant selected by the type, *`CurrentType`*. If the variable does not exist in that branch or the variant is in the non-typed state, then an error results. This error is an exception to the rule that a “set” can set the current type of an IAVariant.
- Setting a typed IAVariant to a non-typed IAVariant sets the type, and then sets the variable.

Assignment of an IAVariant

The code, `p.X.V1 = p.Y.V2`, copies all the values from V1 to V2 provided V1 and V2 are IAVariants of the same type. Otherwise an error occurs.

The "Using IAValueGet/Set Methods with Objects and IAVariants" on page 578 section contains a more in-depth code sample using Objects and IAVariant.

Using MDF Value Attributes

In addition to specifying the data type of each *MDF* value, each data type can have a number of attributes that immediately follow the data type. Separate the attributes with commas. Most modules use the default attributes and specify the input or output attribute as appropriate.

The following codes is an example of how to set an *MDF* value attribute:

```
Object Employee
    Name as String, Output
    JobTitle as String, Output
    Company as String, Const="MyCompanyName"
    Department as Long, Const=27
End Object
```

The "MDF Value Attributes" on page 549 section contains a list of *MDF* value attributes.

Using MDF Value Levels

An *MDF* can include a level in parentheses immediately following the module name. If the triggering level of the module can vary, use the letter `T`, a special character, to substitute for the trigger level number. For example:

```
Module Coding (0)
    InputImage as File,Input     'input image for file
    Keywords as String,Output    'key words on this page
    Importance as Integer,Output 'relative importance of this page
End Module

Module Coding (T)
    Docname as String*20,Output  'document name
    Summary as String,Output     'document summary
End Module
```

This module definition triggers after all of the input images are created. The operator can then specify:

- Keywords and relative importance for each page.
- A document name and summary for each document.

The level of the document is typically 1 or greater. The *System Overview* contains more information on level triggers.

Using MDF Default Values

An *MDF* can include a default value after any *MDF* value. The default value is defined as follows:

```
Module Custom
    StrValue as String, Output = "Default value"
    LongValue as Long, Output = 12345
```

```
    LongValue2 as Long = 23456
End Module
```

Process Developer parses the default values in *MDF*s and add code to the `PreInstall` event handler to set the default values. The default values are stored in special nodes that are not present in the tree. Whenever a new node is created, it inherits its IA Values from the same-level node that has the default values.

The `Process_Install` event handler and the `Batch_Create` event handler can also modify the default values for *MDF* values.

8.6.1.9.4 Inserting MDF Comments

Comments in an *MDF* begin with a standard apostrophe ('). Process Developer ignores everything from the apostrophe to the end of the line.

Here are some examples:

```
' This is a comment at the beginning of a line
```

```
Object CustomerObject
' This comment might describe what the object is used for.
```

8.6.2 Working with IPPs

This section explains how to create, modify, and debug *IPP*.

8.6.2.1 Creating an IPP

To create an *IPP*:

1. From the **File** menu, select **New Project**. The **Define Steps** window displays.
2. In the **Define Steps** window, define steps as instructed in "Modifying a Step in an IPP" on page 502.
3. When you have finished defining steps in the *IPP*, click **OK**.
4. In the **Save As** window, specify a folder and file name for your project.
5. Click **Save**.

8.6.2.2 Opening an Existing IPP

To open an existing *IPP*:

1. From the **File** menu, select **Open Project**.
2. In the **Open** window, specify the folder and file name of the project *IPP* file.
3. Click **Open**.

8.6.2.3 Making a Copy of an IPP

All of the code for an *IPP* is contained in the *IPP* file, including copies of all *MDF*s used in the project. Therefore, copying an *IPP* is as simple as making a copy of the file.

8.6.2.4 Modifying a Step in an IPP

To modify a step in an *IPP*:

1. From the **File** menu, select **Open Project**. The **Open** window displays.
2. In the **Look in** list, find the directory containing the *IPP* that you want to modify.
3. Select the *IPP*, and then click **Open**. Process Developer displays the selected *IPP*. If you modify an *MDF* included in an *IPP* without updating this *MDF* in your project, Process Developer displays an error message. The "Updating an IPP with a Changed MDF" on page 484 section contains instructions on how to handle this error.
4. From the **File** menu, select **Modify Steps**. The **Modify Steps** window displays.
5. To modify the name, module, trigger level, or step level departments of an existing step, select the step, and then select **Modify**.
6. To add a new step, click **Add**.
7. To change the order in which the steps display in the **Modify Steps** window and in Intelligent Capture Administrator, select a step, and then click **Move Up** or **Move Down**. Process Developer rearranges steps accordingly.
8. To delete a step, select a step, and then click **Remove**. Process Developer deletes it from the **Modify Steps** window.

Caution

When you delete a step, you also delete all of the event handlers and code contained in that step.

9. Click **Options** to display the **Options** window. Use the **Options** window to specify whether tasks must automatically be retriggered when the tree is manipulated. The default is to retrigger tasks automatically. Select **OK** when you are done to return to the **Modify Steps** window.
10. Select **OK** when you are finished editing steps.
11. From the **File** menu, select **Save <***IPP_name***>**, where <*IPP_name*> is the name of the *IPP* you were modifying.

8.6.2.5 Converting a PCF to an IPP

Intelligent Capture no longer supports the use of *PCF*s. Process Developer includes a *PCF* import feature that you can use to convert your old *PCF*s to *IPP*. This section explains how to make the conversion.

8.6.2.5.1 Importing a PCF

To import a PCF:

1. In Process Developer select **Import PCF**from the **File** menu.
2. In the **Open** window, specify the folder and file name of the *PCF* you want to convert.
3. Click **Open**.
4. In the **Save As** window, specify a folder and file name for the new *IPP*.
5. Click **Save**.

Steps are contained in the project **Step Objects** folder.

Non-step data is added to Visual Basic modules in the **Modules** folder. The **Main** module contains subroutine code that does not belong with any step, such as *PCF* header information. Constants (such as `IA_ERR_CANCEL`) are included in the **Common_Constants** module. If you included `BAS` file types from within the *PCF*, then they are contained in their own VBA modules and named using the `BAS` file name.

The "Updating Selected PCF Code" on page 503 section contains instructions for making manual code changes necessary for the process to work with the new version of Intelligent Capture.

8.6.2.5.2 Updating Selected PCF Code

When converting a *PCF* to an *IPP*, it can be necessary to make some of the manual code changes described in this section.

8.6.2.5.3 Updating PCF Constant Declarations

When you convert a *PCF* to an *IPP*, constants defined in *MDF*s are moved to the **Main** module of the *IPP*. The event handlers of each step are in separate VBA code modules. Add the `Public` keyword to your constants to make them visible from your event handlers.

For example, when importing a *PCF* that uses the Multi module, the constant declarations are like the following in the **Main** module (this listing is partial):

```
Const IAMULTI_READY = 8          ' do nothing on this node
Const IAMULTI_DELETE = 16        ' delete this node
Const IAMULTI_BEEP = 32          ' beep when processed
Const IAMULTI_PARENT = 64        ' execute on the parent, not each node
```

Change those lines by adding the `Public` keyword:

```
Public Const IAMULTI_READY = 8    ' do nothing on this node
Public Const IAMULTI_DELETE = 16 ' delete this node
Public Const IAMULTI_BEEP = 32    ' beep when processed
Public Const IAMULTI_PARENT = 64 ' execute on the parent, not each node
```

8.6.2.5.4 Updating PCF Tree Navigation Code

If you used tree navigation properties in your original *PCF*, modify the code you converted to an *IPP*. The syntax of tree navigation properties is different in *IPP*. They no longer return error values.

The types of changes you must make are divided into two categories:

- **Changes for properties that loop through nodes**: `Child` and `Page`.
- **Changes for properties that reference only one node**: `NextInLevel`, `NextPeer`, `Parent`, `PrevInLevel`, `PrevPeer`, and `Root`

Each of the following examples demonstrates the old *PCF* syntax (*SBL*) and the new *IPP* syntax (VBA).

To update the syntax of properties that loop through nodes, use the following example as a guideline:

```
SBL: iRet = p.Tree.Child(index,pChild) VBA: Set pChild = p.Tree.Child(index)
```

To change the syntax of properties that do not loop through nodes, use the following example as a guideline:

```
SBL: iRet = p.Tree.NextInLevel(pPeer) VBA: Set pPeer = p.Tree.NextInLevel
```

Note: The syntax changes shown here can also change "Updating PCF Error Handling" on page 506.

If *PCF* code loops through nodes, then change the number of loops executed. For example, the following code is valid in a*PCF*:

```
Sub ImageExp_Finish (p as IAS_RECORD_7)
    .
    .
    .
    iRet = p.Tree.Page(0, pPage)
    iRet = pPage.Tree.Parent(pFold)
    iRet = pFold.Tree.Parent(pStack)
    lNumStacks = p.Tree.NumChildren(2)
    for iStack = 0 to lNumStacks - 1
        .
        .
        .
        iRet = pStack.Tree.NextInLevel(pStack)
        iRet = pStack.Tree.Child(0, pFold)
        iRet = pFold.Tree.Child(0, pPage)
    next iStack
    .
    .
    .
End Sub
```

In this example, when `pStack = lNumStacks - 1`, there is no `NextInLevel`. A *PCF* throws an error when `pStack` is set to nothing. This error occurs as long as `pStack` is

set on the last pass through the loop. Also, it keeps occurring as long as it does not try to use the non-existent `pStack` in the next pass. VBA throws an error, however, as soon as `pStack` is set to the non-existent stack.

8.6.2.5.5 Updating PCF IA Value Methods

If you used methods to set and retrieve IA Values in your original *PCF*, modify the code that you converted to an *IPP*.

Updating Get Methods

The syntax of the `IAValueGetLong` and `IAValueGetAscii` methods has changed—they no longer return error values. The following example demonstrates the old *PCF* syntax (*SBL*) and new *IPP* syntax (*VBA*):

```
SBL: iRet = IAValueGet[Long]/[Ascii]("...",val,12)
VBA: val = IAValueGet[Long]/[Ascii]("...",12)
```

With the new VBA syntax, you can use `IAValueGet` methods in expressions such as:

```
val = 99 * IAValueGetLong("...",12)
str = IAValueGetAscii("...","hi") & " there"
```

Updating Set Methods

The syntax of `IAValueSetLong` and `IAValueSetAscii` has changed. The methods no longer return errors; they throw errors instead. The following example demonstrates the old *PCF* syntax (*SBL*) and new *IPP* syntax (*VBA*):

```
SBL: iRet = IAValueSet[Long]/[Ascii](key, value)
VBA: call IAValueSet[Long]/[Ascii](key, value)
```

Note: The syntax changes shown here can also change "Updating PCF Error Handling" on page 506.

8.6.2.5.6 Updating PCF Procedure Calls

If you passed IA Values by reference to a procedure in your *PCF*, modify your converted *IPP*. In Visual Basic 6, properties of an object are always passed by (`ByVal`), even if the `ByRef` keyword is used.

The "Passing IA Values to Procedures" on page 523 section contains instructions for handling this behavior.

8.6.2.5.7 Updating PCF Error Handling

If you used error handling code in your original *PCF*, modify the code you converted to an *IPP*.

The *SBL* `Err` statement and `Err` function are not used in Visual Basic. If you used them in your original *PCF*, modify your code to use the Visual Basic `On Error` statement and `Err` object instead.

In a *PCF*, the tree navigation properties and `IAValueGet/Set` methods each returned an integer value (typically called `iRet`). The value would be negative if the function was not successful. In your process code, you would be able to check this return value to determine if it was negative and take the appropriate action. The value that the function was designed to get was returned as a parameter.

When using the Intelligent Capture Server, tree navigation properties return the value described by the name of the property. These return values do not indicate an error at all. These properties (as well as `IAValueGet/Set` methods) no longer have return values. Therefore, errors must be handled in other ways, such as by checking the `Number` property of the `Err` object or using the `On Error…` statement. This error handling process makes it more difficult to have code that ignores errors.

The most common forms of the `On Error...` statement are:

- `On Error GoTo` line: Enables the error-handling event that starts at line specified in the required line argument. The line argument is any line label or line number. If a runtime error occurs, then control branches to line, making the error handler active. The specified line must be in the same procedure as the `On Error` statement; otherwise, a compile-time error occurs.
- `On Error Resume Next`: Specifies that when a runtime error occurs, control goes to the statement immediately following the statement where the error occurred where execution continues. Use this form rather than `On Error GoTo` when accessing objects.
- `On Error GoTo ( )`: Disables any enabled error handler in the current procedure.

In addition to these changes, modify your code if you used the `Err` statement and `Err` function. These are *SBL* functions that are not supported in Visual Basic. If you used these functions in your original *PCF*, then we recommend that you change your code to use the Visual Basic `Err` object instead.

Microsoft documentation contains more information about `On Error...` statements.

8.6.2.6 Connecting to an Intelligent Capture Server

For some development tasks such as installing a process or debugging a batch based on a process, log in to an Intelligent Capture Server.

To log in to the Intelligent Capture Server:

1. Launch Process Developer.
2. From the **Tools** menu, select **Connect to Server** to display the **Intelligent Capture Server Login** window.
3. Type the server name and login credentials, and then click **OK**.

8.6.2.7 Compiling and Debugging a Process

After you have created an *IPP*, you can compile, install, configure, and debug it using the Process Developer module. Process Developer includes the Microsoft VBA debugging environment that you can use to debug batches derived from *IPP*.

8.6.2.7.1 Compiling an IPP

Compiling an *IPP* turns it into an *IAP*, which you can install on the Intelligent Capture Server. Then, you use the installed process to configure settings in individual modules. You can compile an *IPP* from the Process Developer *IDE* user interface or from the command line.

To compile an *IPP* using the *IDE*:

1. Create an *IPP* in Process Developer.
2. From the **File** menu, select **Make IAP**. The **Save As** window displays.
3. From the **Save in** field, select the directory in which you wish to save your new process file.
4. In the **File Name** field, type a name for the process.
5. Click **Save**. Process Developer alerts you of any syntax errors or undefined variables in the *IPP*. Fix errors as necessary.
6. Log on to the Intelligent Capture Server and install your process.

To compile an *IPP* from the command line:

- To compile an *IPP* from the command line, use the following syntax:

```
iapdev[ /make][ /NoGUI][ /UpdateMDFs] <filename>.ipp
```

  Where:

  - `iapdev` is the executable name of Process Developer.
  - `/make` is an argument that tells Process Developer to compile the specified <*filename*> *IPP*. Use this argument only if you are using `/UpdateMDFs`.

- `/NoGUI` is an optional argument that suppresses console output and sends it instead to a file named `IAPDev.out`. This file is not deleted or cleared before each compile. If you want it to contain only the results of the current build, manually delete it.
- `/UpdateMDFs` is an optional command-line argument. It checks for newer versions of *MDF*s in the process directory than the ones that are included in the source *IPP* and compiled *IAP* files. If specified with the `/make` argument, it updates the *MDF*s in the compiled *IAP*file only. If specified without the `/make` argument, it updates the *MDF*s in the source *IPP* file only. To update *MDF*s in both the source and compiled files, run `iapdev` twice. Run it once with and once without the `/make` argument.
- *<filename>*`.iap` is the name of the source (*IPP*) file to compile or update. If you specified the `/make` argument, then Process Developer creates *<filename>*`.ipp`.

8.6.2.7.2 Installing an IAP

After you have created and compiled your process and logged on to an Intelligent Capture Server, you can install the resulting *IAP* file using Process Developer. You can also use Intelligent Capture Administrator to install processes. For instructions, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

To install a process on the Intelligent Capture Server using Process Developer:

1. Create an *IAP*, and then log on to the Intelligent Capture Server.
2. From the **File** menu, select **Install Process**. The **Install Process** window displays.
3. In the **File name** field, type the name and path of the *IAP* file that you want to install on the Intelligent Capture Server. This *IAP* was compiled in "Compiling an IPP" on page 507. Or, select **Browse** to search for the file.
4. In the **Process name** field, type a name for the process as you want it to appear in Intelligent Capture Administrator. Type a name which you did not already use for a process or batch on the Intelligent Capture Server. Process Developer does not allow you to overwrite an existing file during installation of a process. For the maximum length allowed, see *OpenText Intelligent Capture - Operating Specifications (ECPCORE-RLI)*. The process name can contain the following characters:
   - A-Z, a-z, 0-9, <space>
   - `- _ ' ' ~ # ! @ $ % [ ] { } ( ) +`
5. Select **Install Process**. Process Developer installs the *IAP* file on the Intelligent Capture Server.
6. Debug your process. The "Debugging a Batch" on page 509 section contains instructions.

Note: You can run multiple Intelligent Capture Servers of different versions. In this case, do not install a process compiled with a newer version of Process Developer onto an older version Intelligent Capture Server. For example, do not install a process compiled with Process Developer version 6.5 onto a version 5.3 Intelligent Capture Server. In cases where the functionality of compiled processes has been changed the Intelligent Capture Server does not allow you to install a newer-version process.

8.6.2.7.3 Configuring Debug Options

Debugging a process that is located on the Intelligent Capture Server pauses the Intelligent Capture Server service. When this service is paused, Process Developer disconnects and stops debugging. You can control whether the service can be paused by modifying the Intelligent Capture Server setting, `CanPauseWhileDebugging`, in Intelligent Capture Administrator.

8.6.2.7.4 Debugging a Batch

In addition to using Process Developer to develop, compile, and install processes, you can also use it for debugging. Process Developer is a fully interactive debugger. It enables you to set breakpoints and step through code. You can also add new events and modify existing events, view and modify values and properties, and receive standard Visual Basic-style error messages during debugging.

To debug a process in Process Developer, create a batch with the process to debug, then run the batch in debug mode. You debug batches because you cannot directly run a process in a production environment. When you run the batch during debugging, Process Developer acts as a server and executes all step events. When debugging, you can run modules on different machines. After you have debugged the batch, you can save any changes to the batch. You can also save the changes to a new *IPP*. Or, you can overwrite an existing *IPP* to capture the changes in a process for future use.

Before you can debug your process, create a batch based on the process. You can then debug the batch.

To debug a batch in Process Developer:

1. Create a batch based on the *IPP* you want to debug. (Each specific module contains information about creating a batch.)
2. From Process Developer, log on to the Intelligent Capture Server ("Connecting to an Intelligent Capture Server" on page 507 contains instructions).
3. From the **File** menu, select **Open Batch**. The **Intelligent Capture Document List** window displays.
4. Select the batch you want to debug and click **OK**. Process Developer opens the batch and displays its project.

5. Set breakpoints, examine variables in the **Watch** and **Local** windows, and use other standard Visual Basic tools to debug your batch.

   Note: To debug your batch, set breakpoints in your code. For example, use the `Image2` process to create a batch. Open the batch in Process Developer, then set a breakpoint at `p.IE.InputImage = p.Scan.OutputImage` in the `Scan_Finish` event handler. Next, begin importing images into ScanPlus. Process Developer stops processing at the breakpoint in your code. You can then single step through your code.

6. From the **File** menu, select **Save** <*Batchname*>. Process Developer saves changes to the batch. It displays a message that changes were saved to the batch, and that you must make any desired changes to the *IPP* separately.

7. From the **File** menu, select **Save As**. The **Save As** window displays, which you can use to save batch changes back to your *IPP*.

   Note: A batch level value named <*Compile_TriggerState*> is used to save the state of the automatic retriggering option when saving an *IPP* from a batch.

8. Type the name and directory of the *IPP* (new or pre-existing) to which to save your changes, then select **Save**. Process Developer saves the changes you made in the batch to the *IPP*.

9. From the **File** menu, select **Close Batch**. Process Developer closes the batch.

10. Compile your process into an *IAP*, and then reinstall it on your Intelligent Capture Servers.

Debugging Restrictions

Debugging in Process Developer is subject to the following restrictions:

- You debug a batch by running only one instance of Process Developer at a time.
- You cannot add or modify steps during debugging.
- You cannot change which IA Values are triggers during debugging. Using a previously unused trigger IA Value does not cause that IA Value to become a trigger. If you remove all references to a trigger IA Value, then it continues to be a trigger. To change which IA Values are triggers, recompile and reinstall the process. ("Working with Trigger IA Values" on page 527 contains instructions.)
- If you get a syntax error while debugging, then do not dismiss the **Waiting for response...** message box. If you ignore syntax errors and continue processing, the module displays a **Waiting for response from server** message box. Selecting **Cancel** in this message box can cause the module to encounter a fatal exception and close. The correct procedure is to dismiss the original syntax error before continuing with your processing.
- If the Intelligent Capture Server is paused during debugging, then Process Developer disconnects and stops debugging. If you set the

`CanPauseWhileDebugging` server parameter to `TRUE` (default setting), then you can pause the Intelligent Capture Server at any time. If you set this value to `TRUE` and the Intelligent Capture Server is paused during debugging, then Process Developer disconnects and debugging stops. To prevent debugging from stopping, set `CanPauseWhileDebugging` to `FALSE`. For instructions about modifying server settings, see*OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

- The Intelligent Capture Server reruns tasks that are stopped during debugging. During debugging, a task can be stopped in the following cases:
  - If you select **Reset** from the **Run** menu
  - If you modify the code and force it to stop
  - If Process Developer stops while executing an event handler.

  If a task is stopped, then the Intelligent Capture Server re-executes it as if the debugger were not present.

  Note: During debugging, changes take effect immediately, which can cause errors if the Intelligent Capture Server stops a task and runs it again. For example, Process Developer executes half an event handler and then stops, and some changes have been made to the batch. Then, these changes can affect how the Intelligent Capture Server runs the batch next time. For example, an event handler can increment an IA Value before Process Developer stops. Then, when the Intelligent Capture Server runs the task again, the IA Value is incremented again. Thus, the IA Value is incremented twice instead of once.
- Process Developer does not automatically save any of the changes to your batch that you make during debugging to your original *IPP* file. Use the **Save As** command to copy them to the *IPP*, and then recompile and reinstall the process on theIntelligent Capture Server.
- After debugging a batch, copy any changes before closing the batch in Process Developer. The Intelligent Capture Server continues processing after you close the batch. If the batch is automatically deleted (for example, by the Multi module) or if you manually delete it upon finishing processing, all debugging changes are lost.

8.6.2.8 Understanding IPP Programming Concepts

This section contains information about writing code for an *IPP*.

8.6.2.8.1 Programming in Visual Basic

Process Developer contains a complete *VBA* programming environment. It simplifies programming tasks by displaying a list of object members that you can choose from as you type your code.

The structure and syntax of your *IPP* must follow the Visual Basic rules. These rules are documented in Microsoft documentation.

Here are some things to keep in mind when you read documentation:

- A Visual Basic module is not the same as an Intelligent Capture module as declared in an *MDF*. Modules declared in *MDF*s actually appear as **Step Objects** in the Visual Basic Editor.
- Some Visual Basic features are not supported in an *IPP*.

8.6.2.8.2 Using Standard VBA Code

Although most standard Visual Basic code in your *IPP* is valid, a list of any limitations is in the "Unsupported Visual Basic Features" on page 550 section. Access the Microsoft documentation from the **Help** menu in Process Developer.

Importing VB Code from Files

You can import new Visual Basic modules and class modules into your project. For example, the Common_Constants module is an included copy of the `aesconst.bas` file.

To import a VB module or class module, select **Import File** from the **File** menu to display the **Import File** window. Only `BAS` VB files are supported. Imported files are added to the modules section.

Note: Any functions, subroutines, and constants in your modules must be declared `Public` if you want to access them from step event handlers or from other modules.

Related Topics

"Using MDF Value Levels" on page 500

"Using MDF Default Values" on page 500

8.6.2.8.3 Understanding How an IPP Works

A completed process is compiled into an *IAP* file that can be installed on an Intelligent Capture Server. Each time an administrator or an operator performs an action with the process (or with a batch created from the process), an event is triggered. At that point, the code inside the appropriate *IPP* event handler determines what, if anything, happens automatically.

This process controls the flow of data through the system. One of the key events associated with each module step is the `Finish` event, within which you can specify what happens with the module output. For example, a typical `Finish` event for a step of the ScanPlus module can be to pass scanned images to the Image Processor module. This module enables operators to clean up scanned images for further processing. For example:

```
Private Sub scanplus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.cpimgpro.InputImage = p.scanplus.OutputImage
End Sub
```

A number of other events are also available, allowing you to control various aspects of the capture process from start to finish.

8.6.2.8.4 Using Common_Constants and Methods in an IPP

When you create an *IPP*. Process Developer adds a Visual Basic module called `Common_Constants` (`aesconst.bas`) to the project. These items are declared publicly so that you can access them from anywhere in the *IPP*. This module includes declarations of the following items:

- Constants intended for use with specific *IPP* features, such as Windows event types for IALogNTMessage.
- Methods for getting and setting fields and named data properties within delimited strings.

The "Methods" on page 576 section contains more details about these common methods.

8.6.2.8.5 Working with Event Handlers

This section explains how event handlers work, and how they can be used to control the flow of information in your process.

Understanding Event Handlers

After defining steps in your *IPP*, you can add code for event handlers. Event handlers comprise much of the code for most *IPP*. The *IPP* waits for events from the Intelligent Capture Server, then uses those events to determine how the batch flow must proceed.

Event handlers are useful for various purposes, including the following:

- Routing tasks (data and images) from module to module

- Validating changes to data
- Recording a log of changes to the batch structure
- Setting default value data for IA Values.

Most event handlers are defined in the *IPP* (such as, `Private Sub EventHandler`). The only exceptions are the `Retrigger` and `Notify` event handlers, which are actually functions—they return values. For this reason, the event handlers are often referred to as routines (for example, the `Finish` routine).

The following are main types of events that your process can use:

1. Tree manipulation event handlers: At each level in the tree (except level 7), there are four tree manipulation event handlers that can be defined:
   - "PostNodeAddn Event" on page 562
   - "PostNodeMoven Event" on page 563
   - "PreNodeDeleten Event" on page 566
   - "PreNodeMoven Event" on page 567
2. Per-step event handlers: There are five event handlers that you can add to any module step in your *IPP*.
   - `Finish` event handlers
   - `Prepare` event handlers
   - `Error` event handlers
   - `Notify` event handlers
   - `Retrigger` event handlers
3. `Batch` and `Process` event handlers: There are three event handlers for batch and process events that you can add to your *IPP*.
   - `Install` event handler
   - `Create` event handler
   - `Delete` event handler

Caution

Do not include long-running operations, such as time consuming complex data validation, calls to external databases or web services, or other complex processing in your *IPP*. All *IPP* code runs in-process and consumes Intelligent Capture Server threads. Such operations can cause all Intelligent Capture Server processing to suspend until they complete, preventing processing of all batches on the Intelligent Capture Server throughout their duration. As a general guideline, limit `Prepare()` code to initializing batch node variables. Limit `Finish()` code to moving or setting batch node data and modifying trigger conditions.

The .NET Code module is aimed at complex custom code. Complex code includes database access, external content server or file server access, complex validations, or data export to proprietary systems. Client-side scripts that are defined in the .NET Code module run outside the Intelligent Capture Server, freeing the server to process batches and schedule tasks efficiently.

Inserting an Event Handler

To insert an event handler in an *IPP*:

1. In the **Project Explorer** window, double-click the step object for which you want to insert an event handler. The **Code** window displays the code for that step object.
2. From the **Code** window's **Object** list (top left), select your step name. (By default, **General** is selected.)
3. Process Developer automatically selects an event from the **Procedures/Events** list (top right). If you are editing a module step, for example, it selects the `Finish` event.

   To insert a different event handler, select it from the **Procedures/Events** list.

   The event handler opening and closing statements (`Sub` and `End`) are automatically inserted into the **Code** window. The default node objects are also automatically declared as arguments.
4. Type code to handle the event between the opening and closing statements.

   You can access the node steps by their argument names (`p`, `p1`, `p2`, and so on).

   Here is an example of a typical `Finish` event handler:

```
Private Sub iascan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.iaipi.InputImage = p.iascan.OutputImage
End Sub
```

Handling Tree Manipulation Events

The tree manipulation events provide the *IPP* with a way of knowing when and how the tree structure has been modified. These events have various uses. For example, they can be used to record information about who is changing the tree. These routines can also be used to duplicate data when you split documents and save data from deleted nodes. If you **Manual retriggering** is enabled, then you can retrigger certain tasks affected by the changes to the tree.

Depending on how you want to modify the tree structure, you can call one of the following tree manipulation events:

- To move one or more nodes, call a `PreNodeMove<n>` event just before the move. Call a `PostNodeMove<n>` event just after the move. No events are generated for the children nodes.

- To delete one or more nodes, call a `PreNodeDelete<n>` event just before the nodes are deleted. No events are generated for the children nodes.
- To insert a node, the situation can be more involved. If the new node is a page node, call a `PostNodeAdd0` event. If the new node is level 1 or higher, and it does not split any other nodes, call a `PostNodeAdd<n> event`. However, if you split a level 1 or higher node when you insert another level 1 or higher node, then multiple events are called. Be sure to note the event order when splitting a document, as it affects how you must write your code.

Event Order when Splitting a Document

The order in which tree manipulation events are called reflects the order of operations that the Intelligent Capture Server performs when it manipulates the tree. Splitting a level 1 node into another level 1 node does not automatically move the pages into the new level 1 node. First, a new level 1 node is inserted after the level 1 node that is being split. Then the pages that were after the split point are moved into the new level 1.

Note: After the new level 1 node is inserted, the pages after the split point are still in the previous level 1 node. Therefore, the new level 1 node has no children. In other words, when the `PostNodeAdd1` event is called, `p1.Tree.NumChildren(0)` is 0. The pages after the split point are not moved into the new level 1 node at the time the `PostNodeAdd1` event is called. Therefore, you must until the `PostNodeMove0` event before retriggering any tasks.

Extend the process by inserting a level 2 node. If this node is inserted between two pages, then a new level 1 node is added after the level 1 node containing those pages. Then the pages after the split point are moved into the new level 1 node. Then a new level 2 node is inserted after the level 2 node that is being split. The level 1 nodes after the split point are moved into the new level 2.

The order in which the `PostNodeAdd<n>` events are called does not necessarily represent the order of the node IDs of the new nodes. For example, a level 2 node is inserted between two pages. The ID of level 2 node can be 471, and that of level 1 can be 472 although the `PostNodeAdd1` event was called first.

Using the pInfo Argument

Using the `pInfo` argument with tree manipulation events allows you to pass information about what module, step, and user caused the event. Passing information is useful to create a log of tree manipulations or take appropriate actions based on who is manipulating the tree.

Handling Task-driven Events

Within each step, there are two events that occur for every task that is processed. When a task begins processing, the `Prepare` event handler is called. When a task completes processing (or aborts), then either the `Finish` event or the `Error` event is called.

The `Finish`, `Error`, and `Prepare` event handlers are called depending on the processing of tasks. This sets them apart from the other per-step events (`Notify` and `Retrigger`), which can be called at any time.

All the event handlers are optional. However, most steps have at least a `Finish` event, as it is typically the primary event handler used for flow control.

The following is a list of three task-driven events and when they are called:

- `Prepare`: Called when a module has a task and is ready to process it.
- `Finish`: Called when a module has finished processing a task.
- `Error`: Called when a module fails to process a task.

Handling the Prepare Event

A `Prepare` event handler initializes values in the step and is executed before sending tasks to the module. When a module is ready to receive a task, the Intelligent Capture Server sends a queued task to the module. Just before the task is sent to the module, the Intelligent Capture Server calls the `Prepare` event handler of the task. This event handler enables code in your *IPP* to validate and modify any necessary data before the module actually begins processing. The `Prepare` event is passed a reference to the task node, for use in processing the event.

For example, you could use a `Prepare` event to format indexing data before it is exported. Formatting can include formatting dates, converting strings to uppercase, and so on. Formatting can also be done in an `Index_Finish` event.

Handling the Finish Event

At a minimum, create `Finish` event handlers for all of the module steps in your *IPP*, except the export steps. `Finish` event handlers are commonly used to route the output of one module step to the input of another. With many custom export modules, `Finish` event handlers are also used for error handling. The Intelligent Capture Server executes a `Finish` event handler each time a client module successfully completes a task.

When a task completes successfully, the Intelligent Capture Server calls the `Finish` event handler of the task. The `Finish` event is passed a reference to the task node for use in processing the event. Because the `Finish` event is called after a task completes successfully, it is useful for routing tasks and validating and formatting data.

The primary purpose of the `Finish` event is to send completed tasks to the next module in the process flow. To send completed tasks, trigger the next step in the process flow.

The `Finish` event handler is useful for routing tasks from one step to the next. It is so useful that almost every step in a typical *IPP* has a `Finish` event. An exception to this rule would be for a step that comes at the end of a process flow. Such a step could be an export module or a step of Multi to delete a batch.

Note: Most of the custom export modules do not call the `Error` event handler when a non-critical error occurs. An example of a non-critical error is when the export failed because one of the exported items was incorrectly formatted. The "Special Notes on Handling Custom Export Module Errors" on page 536 section contains information on how to handle these types of errors.

The `Finish` event handler allows a great deal of flexibility in routing documents. You can include code that checks the state of one or more variables to determine to which step the task must be routed. Many of the preinstalled processes use this dynamic routing.

After a task completes, the module cannot modify the task data any longer. The `Finish` event is therefore an ideal location to validate and format data.

Handling the Error Event

An `Error` event handler instructs the Intelligent Capture Server how to handle the unsuccessful task.

When an error occurs during the processing of a task, the `Error` event handler is called. The `Error` event is passed a reference to the task node, for use in processing the event. Because there are various errors that can occur, the `Error` event handler has an additional argument that holds the error code.

The `Error` event handler provides a way for your *IPP* to take appropriate action when an error occurs. For example, send a page to the RescanPlus module when NuanceOCR times out or is unable to process the page due to poor image quality.

If your *IPP* does not include an `Error` event handler, then the Intelligent Capture Server has a default mechanism for handling errors.

Pay special attention to the `IA_ERR_CANCEL` error code (-4526). This code indicates that the operator canceled the task. In most situations, your `Error` event must retrigger the task (without decrementing <*RetriesLeft*>) if the operator canceled the task. Some modules, such as RescanPlus, depend on this behavior.

Note: Some modules never return an error for a task. The "Special Notes on Handling Custom Export Module Errors" on page 536 section contains more information. Other modules always return an error when they receive a task for the first time. The "Special Notes on Handling RescanPlus and Copy Errors" on page 537 section contains more information.

Handling Module-requested Events

For each step, there are two events that can be called for any task-level node at any time:

- `Notify`: Called by a module to inform the *IPP* that data has been changed for this task-level node.
- `Retrigger`: Called by a module to request retriggering of this task.

These events are called at the request of a module (even if the step is a step of a different module).

Handling the Notify Event

The `Notify` event enables a mechanism for modules to inform the *IPP* that data has changed, without the need to finish a task.

The `Notify` event provides modules with a way to run code in the *IPP* without having to finish a task. A module can call the `Notify` function for any module step in any batch, for any task-level node.

Only a module can call the `Notify` event. Design the module to use this function. There is a special client *API* function available to modules called `IABatchCallNotify`.

The `Notify` event solves this problem by allowing a module to call the `Notify` event in place of a `Finish` event on the non-task node. The *IPP* then knows that data was modified, but a task was not completed. The *IPP* could then trigger a module to handle the new data.

Because of the `pInfo` argument that is passed, the *IPP* even knows what module called the `Notify` event, for which step, and by which user.

Handling the Retrigger Event

A `Retrigger` event handler enables your *IPP* to intercept a request to retrigger a task node, and dynamically determine whether to carry out that request.

The `Retrigger` event provides modules with a way to request that the Intelligent Capture Server retrigger individual tasks within a batch. This mechanism enables the *IPP* to run any necessary code before a task is retriggered, including ignoring the request. Thus, this mechanism enables an *IPP* to determine dynamically whether the task needs being retriggered.

A module can call the `Retrigger` function for any module step in any batch, for any task-level node. The "Retrigger Event" on page 568 section contains additional information about the arguments.

Only a module can call the `Retrigger` event. Design the module to use this function. There is a special client *API* function available to modules called `IABatchRetriggerNode`.

Because of the *`pInfo`* argument that is passed, the *IPP* knows what module called the `Retrigger` event, for which step, and by which user.

Handling Process and Batch Events

The `Process_PreInstall` event handler is a fourth type of event handler that Process Developer creates automatically, incorporating default values set in the *MDF*s. However, you can add a `Process_Install` event handler to add any

additional default values and run any other code necessary when the process is installed.

The `Batch_Create` event enables you to set any additional values at the time of batch creation. The `Batch_Delete` event enables you to log information to an external repository before the batch is deleted.

The events discussed so far (task-driven, module-driven, and tree manipulation events) require the users to manipulate the batch in some way: to process a task, use a module to call an event, or manipulate the tree. These events can be called multiple times, depending on how the process flow is set up and how the users process tasks.

The `Process` and `Batch` events are called as follows:

- When you install an *IAP*, the `Process_PreInstall` event is called, followed by the `Process_Install` event. Process Developer automatically generates the `PreInstall` event. Do not attempt to modify it. However, you can use the `Install` event to initialize default values for nodal values (*MDF* values and dynamic values).
- When you create a batch, the `Batch_Create` event is called. You can use the `Create` event to perform any initialization necessary for a batch. Initialization can include setting default values that could not be set when the *IAP* was installed, such as the batch creation date.
- When you delete a batch, the `Batch_Delete` event is called. After the `Delete` event has completed processing, the batch is deleted. This event is useful to log information to an external repository or to keep a log of when the batch was deleted and by whom.

Note: The batch is deleted immediately after the `Batch_Delete` event finishes. Any tasks the *IPP* attempts to trigger are ignored and do not get a chance to run before the batch is deleted. Your *IPP* cannot prevent the batch from being deleted after the `Batch_Delete` event is called.

8.6.2.8.6 Working with Objects

Because Process Developer is built upon Visual Basic technology, it uses an object model for storing information about all the nodes and steps in a batch. The object model allows a wide variety of variables and functions that you must program to consolidate your *IPP* into a few locations.

Working with Step Objects

Each *IPP* has a folder named **Step Objects**, which contains the following:

- **One step object for each module step defined in the project**: Each step object name is prefixed with `Inst<###>_`, where <###> represents the sequential number of the step in the *IPP*. If you change the order of steps, then Process Developer adjusts these numbers accordingly. (The prefixes are not displayed anywhere else in Intelligent Capture.)

These step objects contain the code that runs when the associated module is used on a batch created from the process. All of this code is stored inside the objects event handlers.

- **Batch, Process, and Tree objects**: These objects contain code for events that occur on the associated items, independent of specific Intelligent Capture modules.

Working with Node Objects

Node objects are the top-level containers that you use to access all other objects in your *IPP* code. As the name implies, they represent nodes in the batch tree.

Each event handler has one or more node objects passed to it as arguments. They are named `p`, `p1`, `p2`, `p3`, `p4`, `p5`, `p6`, and `pRoot`, for levels 0 through 7, respectively. The actual data type of a node object is `IASLib.IAS_RECORD_<N>` where <*N*> is the level of the node. It can be referenced as simply `IAS_RECORD_<N>`. The contents of the `IAS_RECORD_<N>` type depend on the steps defined in your process.

For each step, `IAS_RECORD_<N>` contains an object whose name matches the step name. Its data type is `IAS_<INSTANCENAME_N>`. In addition, every node object contains a `Tree` object of type `IAS_TREE_<N>`.

These child objects in turn contain all of the objects and IA Values defined for their associated modules. Those items can be default built-in items, or items defined in an *MDF*.

The following example would set a custom variable named `ScanKeyEntry` to the data stored in the ScanPlus module level 1 IA Value:

```
ScanKeyEntry = p1.scanplus.Level_1_KeyEntry_0
```

Working with Tree Objects

For each level in the node tree, there is a corresponding `Tree` object of data type `IAS_TREE_<N>`, where <*N*> is the level in the tree.

Some `Tree` properties return information about the node, such as the number of children or the node ID. Some properties actually return another node object and can thus be used to navigate the tree. Each `Tree` object exposes properties appropriate for the node level. For example, most Tree objects have a Parent property that returns the parent node. But, the level 7 tree objects do not have this property because this level is the root level of the tree. Thus, nodes at this level do not have parent nodes.

Working with Custom Objects

If you are writing an *IPP* for a process that handles several items of data per page, use objects to improve the processing speed. With an object, you can send many values to the server in one transaction instead of requiring a separate transaction for each value.

Here are some performance tips to keep in mind as you work with custom objects:

- When declaring a value as an object in an *MDF*, use the `Prefetch` keyword to send the object to the module along with the task data. For example:

```
Level0_Object as MAPPING,Prefetch
```

- In the *IPP*, use the object members to hold IA Value data that you want to pass among module steps.
- When setting up the module step, do not select the IA Values from other steps as the settings for the module. Instead, choose the values from the object that you created.

8.6.2.8.7 Working with IA Values

After creating the *IPP*, with the steps and event handlers, the main task remaining is to add the code to control the process flow.

When adding your code, Process Developer gives you access to several types of information about your batch. You can access the tree hierarchy and set IA Values and triggers on any node in the batch. Triggering can be done including on children and parent nodes of the node that is passed by the event handler.

This power and flexibility and made accessible by use of VBA technology. Various objects are provided to the user to expose the structure and information of a batch.

Accessing IA Values

There are three main types of IA Values available within a batch:

- **Nodal IA Values**: These IA Values are associated with a certain level node. For each node in the tree at that level, there is a unique copy of this IA Value. Both *MDF* values and dynamic values are nodal values. You can automatically insert an *MDF* value into your code using the Process Developer auto-fill feature.
- **Per-step IA Values**: These IA Values are typically used for storing step settings. Storing these values in the step, rather than for each and every node, saves considerable space. Because per-step values are not stored on a node, you can only set and get them with the `IAValueGet/Set` methods.
- **Per-batch IA Values**: These IA Values are associated with the batch. Some of these values can be accessed from the node reference objects that are passed to the event handlers (for example, `p.Tree.BatchName`). However, most per-batch IA Values must be accessed with the `IAValueGet/Set` methods.

Adding MDF Values Using Auto-fill

You can automatically insert a tree navigation property into your code by using the Process Developer auto-fill feature.

To insert an *MDF* value into an event handler:

1. Create an event handler, then position the cursor between the beginning and end statements of the event handler.

2. Begin typing using the syntax:

```
p<#>.<StepName>.<ValueName>
```

   Where <#> is the trigger level for the step. When the step is triggered at level 0, you do not need to indicate the trigger level. When it is triggered at level 7, specify `pRoot`. Do not specify a number, in case the numbering of the highest level is ever changed in a future release of Intelligent Capture. When you start to type this syntax, Process Developer displays available *MDF* values within the auto-fill feature.

3. Use the up and down keys to select an IA Value from the list, then press **TAB**. Process Developer automatically fills in the selected value.

Using `IAValueGet/Set` Methods

`IAValueGet/Set` methods retrieve and set batch string and long integer values without first traversing the tree.

To insert an `IAValueGet/Set` method into an event handler:

1. Create an event handler using instructions in "Inserting an Event Handler" on page 515, then insert your cursor between the beginning and end statements of the event handler.

2. Type the method, followed by a beginning parenthesis. Process Developer displays a window with the correct arguments for the method

3. Type the arguments using the syntax displayed.

Process Developer automatically displays argument information for `IAValueGet/Set` methods when you begin typing one of these methods into your *IPP*. By automatically displaying the proper arguments as you type, the pop-up syntax feature simplifies typing `IAValueGet/Set` methods.

Passing IA Values to Procedures

You cannot directly pass an individual IA Value by reference to a procedure. In Visual Basic 6, properties of an object are always passed by value (`ByVal`), even if the `ByRef` keyword is used. The following *IPP* example demonstrates this behavior:

```
Sub ChangeDescription(ByRef desc)
    desc = desc & " - Pages Scanned"
End Sub

Private Sub scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.Tree.Description = "New Batch"
    ' The following procedure call does not work as expected
    Call ChangeDescription(p.Tree.Description)
End Sub
```

After this runs, the value data for `p.Tree.Description` is set to `"New Batch"`. The `ChangeDescription` method does not change the value data.

Here are some ways to work around this behavior:

- **Pass the entire object whose property you want to change:**

```
Sub ChangeDescription(ByRef tree)
    tree.Description = tree.Description & " - Pages Scanned"
End Sub

Private Sub scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.Tree.Description = "New Batch"
    Call ChangeDescription(p.Tree)
End Sub
```

- **Create a temporary variable to pass to the event handler:**

```
Sub ChangeDescription(ByRef desc)
    desc = desc & " - Pages Scanned"
End Sub

Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.Tree.Description = "New Batch"
    tempDesc = p.Tree.Description
    Call ChangeDescription(tempDesc)
    p.Tree.Description = tempDesc
End Sub
```

- **Use a function instead of an event handler:**

```
Function ChangeDescription(desc)

ChangeDescription = desc & " - Pages
Scanned" End Function  Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.Tree.Description = "New Batch"
    p.Tree.Description = ChangeDescription(p.Tree.Description)
End Sub
```

Creating Dynamic Values

With the Intelligent Capture Server, you can create Dynamic Values. Dynamic Values are IA Values of the type String that are created “on the fly”. They are created from an *IPP* or from a client module designed to create them. In modules that do not create Dynamic Values, IA Values must be declared in advance in the Module Definition File. After Dynamic Values are created, they behave almost exactly like *MDF* Values.

Using standard Visual Basic for Applications syntax, you can create a Dynamic Value in an *IPP* by setting the variable. If the variable does not exist, then the Intelligent Capture Server creates it during processing. You do not need to declare the Dynamic Value. You can reference the Dynamic Values to make the Intelligent Capture Server create them.

Notes

- For the maximum length allowed, see *OpenText Intelligent Capture - Operating Specifications (ECPCORE-RLI)*. Characters are restricted to alphanumeric and underscore (_). The first character must be a letter.
- Dynamic Values are not created until runtime, therefore they are not available in Process Developer pick lists. Nor are they available in the setup pick lists for other client modules (excepting batches that have already executed the code that creates the Dynamic Value).

When developing your *IPP*, you can also assign Dynamic Values to different level nodes within the same *IPP* event handler. For example, the following sample code

creates two Dynamic Values. `Level0Variable` is assigned to the string `"My name"` and stored for each level 0 node in the ScanPlus step. `Level1Variable` is assigned the value 120 and stored for each level 1 node in the ScanPlus step.

```
Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    p.Scan("Level0Variable") = "My name"
    p.Tree.Parent.Scan("Level1Variable") = "120"
End Sub
```

Dynamic Value Syntax

To create a Dynamic Value in your *IPP*, use the syntax:

```
p<#>.<Step>("<DynamicValueName> ") = "<Value> "
```

Where:

- <#> indicates the level at which you wish to create the variable.
- <*Step*> is the name of the step in which you want to store the variable.
- <*DynamicValueName*> is the name you want to assign to the variable. (The "Dynamic Value Naming Conventions" on page 525 section contains a list of naming restrictions.)
- <*Value*> is the value to assign to the variable.

When you use this syntax, the Intelligent Capture Server creates the Dynamic Value for each node of the specified level in the specified step for the batch. For example, the following line of code creates a Dynamic Value named `"MyValue"`. This dynamic value stores the String value `10` for each level 1 node in the ScanPlus step for this batch:

```
p1.Scan("MyValue") = "10"
```

You can also create Dynamic Values using the syntax:

```
p<#>.<Step>.Item("<DynamicValueName> ") = "<value> "
```

This syntax functions the same as the previous syntax, except that it includes the object <*Item*>. Use the syntax of your preference. The Dynamic Values that the Intelligent Capture Server creates from both lines of code have identical properties.

Dynamic Value Naming Conventions

Each Dynamic Value is subject to the following naming rules:

- It must have a unique name.
- It must begin with a letter.
- It must contain only letters, numbers, and underscores.
- For the maximum length allowed, see *OpenText Intelligent Capture - Operating Specifications (ECPCORE-RLI)*.
- It must follow standard Visual Basic naming conventions.

For example, the following Dynamic Value names are valid:

```
LastName
LastName2
Last_Name
```

The following Dynamic Value names are not valid:

```
_LastName
2LastName
Last/Name
```

Note: Never use the same name for a variable that is already used elsewhere for an event handler, method, or property. Also, do not use any of the IA Value names listed in "IA Values" on page 570.

Dynamic Value Rules and Restrictions

In addition to the naming conventions, Dynamic Values are subject to the following rules and restrictions:

- Dynamic Values can only be of the data type String.
- You cannot create a Dynamic Value in an *IPP* using `IAValueSetAscii` or `IAValueSetLong`.
- You cannot create a Dynamic Value using an existing name. If you try to create a Dynamic Value that exists in an *MDF* used by the *IPP*, the Intelligent Capture Server uses the existing variable.
- You cannot use a Dynamic Value as a trigger variable. You cannot use a Dynamic Value as a trigger variable in your *IPP* because all triggers, except <*Ready*>, must be defined in an *MDF*.
- You cannot view Dynamic Values created by a process in client Modules setup lists. Client modules can display IA Values defined in an *MDF* during setup mode, but they cannot display Dynamic Values you created in an *IPP*. They cannot display these Dynamic Values because when you set up a process, the *IPP* code that creates them has not yet run. To create variables that client modules can display, use another mechanism to create them, such as adding IA Values to a custom *MDF*.
- You cannot access a Dynamic Value in the *IPP* using the syntax `p.<StepName>.<DynamicValue>`. To access a Dynamic Value in your *IPP*, use the syntax `p.<StepName>("<DynamicValue>")`.

Modifying Departments Dynamically in an IPP

There are two string IA Values that you can set in an *IPP* to control department routing—one for step-level routing and one for task-level routing. The task-level routing IA Value (<*IATaskRouting*>), if set, overrides the step-level routing IA Value (either the <*IADepartments*> value or the static department declaration). If the step-level dynamic value (<*IADepartments*>) is set, it overrides the static department declaration.

- <*IADepartments*>: specifies the current step-level value of the department. Setting this value programmatically reduces the need to use multiple module steps to route tasks to different departments. You can access this IA Value by using the `IAValueGet/Set` methods with the special strings for constructing strKey.
- <*IATaskRouting*>: specifies the current task-level value of the department and overrides any step-level department value that is set. This value is on the current task. Thus, you can set it without using a strKey and test it to determine how to route the task.

Notes

- When departments are defined dynamically, the department names do not appear in the Intelligent Capture Administrator **Departments** pane until at least one task from the first batch is processed though the module step where they are defined. Administrative access to the departments can be required before running a batch. For example, this access is necessary to grant or revoke permissions in department *ACL*s. In this case, you can add the department names manually by using the **Add** button in the **Departments** pane.
- For the maximum length allowed, see *OpenText Intelligent Capture - Operating Specifications (ECPCORE-RLI)*. It is a best practice to define department names that are short, easy to remember, and easy to type correctly in a command line. Short names are easier to identify in the department name lists in the Intelligent Capture Administrator where fields can have limited space.

Working with Trigger IA Values

This section describes how Process Developer determines which IA Values behave as triggers. Trigger IA Values are special IA Values that determine which tasks in a batch are available to send to modules. To make an IA Value act as a trigger, or to enable it, declare it as a potential trigger in an *MDF*. Then reference it in an *IPP*.

The Intelligent Capture Server triggers a task when the task and all of its children have all their trigger values for a step set to non-zero values.

Note: If an IA Value that is declared in an *MDF* as a trigger is used in an *IPP*, then it automatically becomes a trigger. A common error is to use a value without realizing that it is a trigger value. This common error often leads to confusion about what is preventing a task from being triggered.

There are two special IA Values declared by Process Developer: <*Ready*> and <*RetriesLeft*>. The <*Ready*> IA Value is a trigger. If it is referenced in the process, it is automatically declared for each step and level. It follows the same rules for referencing triggers. The <*RetriesLeft*> IA Value is a counter and is automatically declared for each step at its task level. <*RetriesLeft*> is enabled regardless of whether the process includes a reference to it. Although <*RetriesLeft*> is not a trigger, it controls the flow of tasks through the process. By default, the <*RetriesLeft*> Value is set to 3.

All of the *MDF*s included with Intelligent Capture declare triggers, except *MDF*s corresponding with importing and scanning modules. A typical trigger is <*InputImage*>, which is declared in the *MDF*s of modules that use images. Modules that serve as the starting point for a process, such as ScanPlus, do not need trigger values. They are triggered, for example, by the operator selecting the **Scan** button.

You can set a trigger value at a higher level than the trigger level. If you do so, then this higher-level trigger must be satisfied before the batch moves to the next step in your process.

Referencing Triggers

To reference a trigger, and thus enable it, access it through a step object in your process. The access must be through a non-array variable of the type `IAS_RECORD_<n>` or `IAS_<STEPNAME>_<n>`. The access cannot be through the Dynamic Value syntax or methods such as `IAValueGetAscii`. To enable a trigger, simply reference it once from anywhere in your process.

Note: If you do not enable a trigger IA Value, then the value is not used in determining whether a task is ready. In other words, the value acts as though it were not a trigger IA Value.

Here are examples of how to enable a trigger. In the examples,

- <*p*> is an object of type `IAS_RECORD_<n>`.
- <*Step*> is a step defined for the process.
- <*MyValue*> is a possible trigger IA Value defined on the step.

The following is an example of how to enable the trigger, “MyValue”, through the `IAS_RECORD_<n>` object:

```
Private Sub StepName_Finish(ByVal p As IASLib.IAS_RECORD_n)
    p.StepName.MyValue = 1
End Sub
```

The following example enables the “MyValue” trigger through the `IAS_<STEPNAME>_<n>` object:

```
Private Sub StepName_Finish(ByVal p As IASLib.IAS_RECORD_n)
    Dim Step As IAS_StepName_n
    Set Step = p.StepName
    Step.MyValue = 1
End Sub
```

The following example enables the “MyValue” trigger by using a `With` statement:

```
Private Sub StepName_Finish(ByVal p As IASLib.IAS_RECORD_n)
    With p.StepName
        .MyValue = 1
    End With
End Sub
```

Including Additional Triggers in your IPP

When you create an *IPP* using Process Developer and the Intelligent Capture Server, you can include up to 256 triggers in your process. This extended trigger limit gives you more flexibility when creating processes.

Preventing Empty Folders from Triggering

"folder" is a generic term representing an empty level 1 through 6 node. When you create an empty folder, the module steps at that level can trigger when the node is created.

The module steps at the level of the empty node can trigger also when you move all the children out of a folder. For example, you move all the pages out of a level 1 node, or all the level 1 nodes out of a level 2 node.

This triggering situations have to do with the definition of when a task is triggered. A task is triggered when all of its triggers variables are set to non-zero values. Its trigger variables include triggers on the task node as well as triggers on its children nodes.

In many situations, *IPP* writers trigger tasks by using page level triggers, typically an input file (for example, `Export.InputImage`). If a task node has no children nodes, then there are no page level triggers to be set to 0. The only trigger remaining that is enabled by default is the <*Ready*> value. This special trigger value is automatically declared when you reference it in your *IPP*. Setting <*Ready*> to a non-zero value triggers any Intelligent Capture module.

There are two ways to prevent tasks on empty folders from triggering:

- Manual retriggering
- Auto-retriggering

The simplest way to prevent tasks on empty folders from triggering is to enable manual retriggering. When you enable manual retriggering, changes to the tree do not cause the Intelligent Capture Server to re-evaluate the ready status of completed tasks. Changes include:

- **Adding nodes**: For example, creating an empty level 1 node in ScanPlus
- **Moving nodes**: For example, moving all the pages out of a level 1 node from Completion
- **Deleting nodes**: For example, deleting all the pages in a level 1 node from Multi

While manual retriggering successfully prevents tasks on empty folders from mistakenly becoming ready, it can also prevent tasks from retriggering when the tree has changed. Retriggering a task when its subtree has changed can be desirable, as it can enable you to reprocess a task that has changed. That example includes code to prevent empty folders from retriggering.

Note: Use Manual Retriggering so that the server sets the ready status to "Done" after running a Multi step to add, delete, or move a node. In previous releases, the ready status appeared as "Not Ready." To exhibit this old behavior, modify your *IPP* to clear a trigger value from the step that you want to be set to "Not Ready".

To keep automatic retriggering enabled, and prevent empty folders from automatically triggering, create a situation where the task is not ready by default. Create this situation by enabling the <*Ready*> trigger at the task level for the steps that you want to prevent from being ready. This situation is like preventing retriggering when auto-retriggering is enabled.

8.6.2.8.8 Retriggering Tasks

Retriggering consists of sending tasks back to modules that have already processed the tasks. Retriggering can be intentional (such as retriggering a step from Intelligent Capture Administrator), or is not (such as manipulating the tree). There are three main methods of retriggering tasks:

- **Retriggering by manipulating the tree**: Whenever you modify the tree structure, the Intelligent Capture Server re-evaluates the trigger IA Values on every level 1 node and higher. This re-evaluation can retrigger multiple steps at multiple levels at once. The "Retriggering by Manipulating the Tree" on page 531 section contains more information.

  Note: If you set up the *IPP* with manual retriggering, then tasks do not automatically retrigger when you manipulate the tree.

- **Retriggering by setting trigger IA Values**: You can get a task that has completed to be ready again. You can set one of its triggers to a non-zero value. Or, you can set the value to itself. The "Retriggering by Setting Trigger IA Values" on page 532 section contains more information.

- **Retriggering a step from Intelligent Capture Administrator**: For each task node in the tree, the Intelligent Capture Administrator sends a request to the Intelligent Capture Server to retrigger the task. The Intelligent Capture Server then calls the `<StepName>_Retrigger` event handler, which you can provide in your *IPP*, to handle the event appropriately and retrigger the task. If the `Retrigger` event is not present in the *IPP*, then a standard method of retriggering is used. For more information, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

Setting Automatic Retriggering

Within the Intelligent Capture system, retriggering is automatic by default. You can disable automatic retriggering, by configuring Process Developer. To keep automatic retriggering enabled, but prevent automatic retriggering on a step-by-step basis, you can include special code in your *IPP*.

Retrigger the step manually if you want to retrigger it after a page is deleted. In other words, set a trigger variable again.

If you enable manual retriggering, the Intelligent Capture Server sets the ready status to “Done” after a Multi step. This step is run to add, delete, or move a node. In previous releases, the ready status appeared as “Not Ready.”. To exhibit this old behavior, modify your *IPP* to clear a trigger value from the step that you want to be set to “Not Ready.”.

Retriggering by Manipulating the Tree

When a tree structure is modified during processing, the Intelligent Capture Server can retrigger tasks contained within that tree. Retriggering consists of automatically sending tasks back to modules that have already processed the tasks. The Intelligent Capture Server sends the tasks back to make sure that all of the new work generated by the changed tree structure is properly processed.

For example, look at the following structure:

```
Scan (ScanPlus at Level 0)
    |
    |
Hold (IAMulti at Level 7)
    |
    |
IE (IAIPI at Level 0)
    |
    |
Index (IndxPlus at Level 7)
```

Suppose you ran a batch through `Scan`, `Hold`, and `IE` using this process and inserted a level 1 node during `Index` processing. The entire batch would be sent back to `Hold` for reprocessing. It would be sent back because Intelligent Capture Server retriggers tasks when the subtree (that is, nodes within the task) is deleted, moved, or inserted. In this case, the task level is 7 (the batch level). After `Hold` finishes reprocessing, `IE` and `Index` must reprocess as well.

Note: This method can retrigger only the steps that trigger tasks at level 1 or higher.

By default, auto-retriggering is enabled. In this mode, when you change the tree in a batch, the Intelligent Capture Server can retrigger all affected tasks automatically. Affected tasks include any tasks triggered at or higher than the level of the changes in the tree. Automatic retriggering can be avoided by disabling automatic retriggering in the **Options** window and using manual retriggering. To disable automatic retriggering, click the **Options** button in the **Modify Steps** window to display the **Options** window, then select the **Manual retriggering** option.

Retriggering Selected Tasks within the Batch

The example shown in “Retriggering by Manipulating the Tree” on page 531 retriggers the entire batch for the `Hold` step after a modification of the tree structure. The `Hold` step is triggered at the batch level. You can select tasks to retrigger within a batch, depending on the trigger level of your steps and modification of your tree structure.

In the following example, the `Hold` step is triggered at level 2:

```
Scan (ScanPlus at Level 0)
    |
    |
Hold (IAMulti at Level 2)
    |
    |
IE (IAIPI at Level 0)
    |
    |
Index (IndxPlus at Level 7)
```

Suppose you inserted a level 1 node during indexing using this process. Then, only the level 2 node that was affected would be retriggered for the `Hold` step and then reprocessed by the other steps. The unaffected level 2 nodes in the batch would not be retriggered because their subtree structures would remain the same.

To avoid retriggering the `Hold` step, you would need to set one or more `Hold` triggers to 0 in the `Hold_Finish` or `IE_Finish` event handlers. The only difference is that the `Hold_Finish` would take a level 2 node as the argument.

Preventing Retriggering with Auto-retriggering Enabled

If you want to keep automatic retriggering enabled, include special code in your *IPP* to prevent automatic retriggering. The easiest way is to set the <*Ready*> trigger for a step equal to zero in the `Finish` event handler for that step. As a result, the step runs once before it is untriggered.

Preventing Retriggering with Manual Retriggering Enabled

When you change the tree in a batch, all affected tasks can be retriggered. The easiest way to prevent automatic retriggering is to enable **Manual Retriggering** from the **Options** window of the **Define Steps** or **Modify Steps** window. When you enable manual retriggering, changes to the tree do not cause the Intelligent Capture Server to re-evaluate the ready status of completed tasks. To retrigger a task automatically when its subtree is modified, use the tree manipulation event handlers.

Note: If you enable **Manual Retriggering**, then the server now sets the ready status to "Done" after running a Multi step to add, delete, or move a node. (In previous releases, the ready status appeared as "Not Ready". To exhibit this old behavior, then modify your *IPP* to clear a trigger value from the step that you want to be set to "Not Ready".)

Retriggering by Setting Trigger IA Values

When a task is complete, its trigger IA Values can all still be non-zero. If the Intelligent Capture Server has to re-evaluate the status of the task, the task can become ready again. For a task to be ready again, set one of the task triggers to a non-zero value or set the trigger IA Value to itself.

8.6.2.8.9 Handling Errors

There are two distinct types of errors that can occur when running a process:

- Task-level batch errors occur during processing and when a module signals that it cannot properly process the task.
- Unhandled errors, or runtime errors, occur when the *IPP* is running and are due to errors in the code. For example, dividing a number by zero in your *IPP* causes an unhandled error.

The following sections describe Intelligent Capture Server behavior when these errors occur and provide guidelines on how to handle them in your *IPP*.

- "Handling Task Errors" on page 533
- "Handling Runtime Errors" on page 534
- "Special Notes on Handling Custom Export Module Errors" on page 536
- "Special Notes on Handling RescanPlus and Copy Errors" on page 537

Handling Task Errors

When a module error occurs while running a task, the Error event handler for the step of that task is called. When the `Error` event handler is called, it sets the `Code` parameter to the error handling mode that the module returned. If an `Error` event handler does not exist, then the Intelligent Capture Server defaults to the following behavior:

Table 8-2: Intelligent Capture Server Default Error Handling

Module Result	Description
`IA_ERR_NORETRY`	No retriggering occurs. The task is done.
`IA_ERR_RETRYSOME`	If <*RetriesLeft*>> 0, then decrement <*RetriesLeft*> on the task. If <*RetriesLeft*> is still > 0, then the task is retriggered. Otherwise, the task returns an error, and a warning is logged.

`IA_ERR_CANCEL`
`IA_ERR_RETRY`
Any other error
	Retrigger the task.

Note: Some modules never return an error for a task. The "Special Notes on Handling Custom Export Module Errors" on page 536 section contains more information. Other modules always return an error when they receive a task for the first time. The "Special Notes on Handling RescanPlus and Copy Errors" on page 537 section contains more information.

These `IA_ERR` constants are declared in the Common_Constants module as follows:

```
Public Const IA_SUCCESS=0
Public Const IA_ERR_CANCEL=-4526
Public Const IA_ERR_NORETRY=-6112
Public Const IA_ERR_RETRYSOME=-6113
Public Const IA_ERR_RETRY=-6114
```

For a complete list of client module error and log codes, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

The following sample uses a subroutine in the Main code module called `Error_Handler` to write an error event for a step.

```
Public Sub Error_Handler(StepName As String, lNode As Long, _lBatch As Long, iCode As
Integer)
    Dim strBatchname As String
    strBatchname = IAValueGetAscii("$batch=" & lBatch & "/Batchname", "Error")

    Call IALogNTMessage(IALOG_ERROR, "Error in Batch " & strBatchname & " with node
number " & lNode & ". step '" & StepName & "' returned error code " & iCode & "." )
End Sub
```

The following is a sample `IE_Error` event handler which calls the `Error_Handler` subroutine. Use this subroutine in the `Error` event handler of each step. It enables trapping and evaluating the error `Code` in a uniform manner to take appropriate action:

```
Option Explicit

Private Sub IE_Error (ByVal p As IASLib.IAS_RECORD_0, ByVal Code As Integer)
    ' Send an error message to the Windows Event Log:
    Call Error_Handler("IE", p.Tree.NodeID, p.Tree.BatchID, Code)

    ' Note: the following code is the "default" behavior
    ' that would occur if this event did not exist at all.

    ' The following code handles errors by using error
     ' codes and the RetriesLeft value:
    If p.IE.RetriesLeft > 0 Then
        If Code = IA_ERR_RETRYSOME Then
            p.IE.RetriesLeft = p.IE.RetriesLeft-1
        Elseif Code = IA_ERR_NORETRY Then
            p.IE.RetriesLeft = 0
        Else
            p.IE.RetriesLeft = p.IE.RetriesLeft
        End if
    End If
    ' As an option, you can set the Batch priority
    ' to 0 if the RetriesLeft of a step goes to 0.
    'If p.IE.RetriesLeft = 0 Then
    ' Call IAValueSetLong("$batch=" & p.Tree.BatchID & "/Priority", 0)
    'End If
End Sub
```

Handling Runtime Errors

If an unhandled, or runtime error occurs, the Intelligent Capture Server, sets the batch priority to 0 and logs a warning event to the server. The warning event is equivalent to the Intelligent Capture Server calling `IALogNTMessage` with `iType =IALOG_WARNING`. Due to restrictions in the VBA language, the Intelligent Capture Server does not inform you about the type, state, or line of the error.

If you wish to handle runtime errors differently in your *IPP*, it is recommended that you include `On Error GoTo...` statements in the event handlers that are most likely

to encounter runtime errors (specifically, statements that perform arithmetic operations or string manipulation). Then, post your error messages outside the Intelligent Capture environment (such as the event log). This way you can read the error message even if the Intelligent Capture Server is in an invalid state.

Note: When you trap your own runtime errors, your code overrides the Intelligent Capture Server error handler functions within the event in question. The Intelligent Capture Server assumes that errors are handled appropriately, and if mishandled, a message does not appear in either the `Debug.out` file or the event log.

It is recommended that you construct error messages that include the error type and the line on which the error occurred. For example:

```
"A run-time error 6 (Overflow) occurred on line 55."
```

The VBA language does not include the ability to retrieve line numbers. To get the global function `Erl()` to work correctly, add line labels manually to your code. A line label identifies a single line of code. A line label can include any combination of characters that starts with an alphanumeric character (or string) and ends with a colon (:). Line labels are not case sensitive and must begin in the first column.

Note: You cannot use third-party software packages with Process Developer that add line numbering to a VBA project. Also note that when you use the line continuation operator "_" you cannot label the next line. Doing so results in an "Expected end of statement..."" error message.

The following example includes line labeling and error handling.

```
Option Explicit

Private Sub Index_Finish(ByVal p As IASLib.IAS_RECORD_0)
1:    Dim x As Integer
2:    Dim szErrMessage As String
3:
4:    On Error GoTo Err_Handle
5:
6:    x = 1
7:
8:    x = (x - 1) / 0 ' Divide by zero - always error
9:
10:    Exit Sub
11:
12:        Err_Handle:
13:
14:    szErrMessage = "A run-time error " & Err.Number & " (" & Err.Description & ")
occurred in batch "
         & p.Tree.BatchName & " within the Index_Finish on line " & Erl() & " with node
" & p.Tree.NodeID
         & ". The priority of this batch will be set to 0 to prevent further problems."
18:
19:    Call IALogNTMessage(&H1, szErrMessage)
20:
21:    ' Set a message in the description box of the batch
22:    Call IAValueSetAscii("$batch=" & p.Tree.BatchID & "/Description", szErrMessage)
23:    ' Set the priority of the batch to zero
24:    Call IAValueSetLong("$batch=" & p.Tree.BatchID & "/Priority", 0)
25: End Sub
```

When using the previous sample, runtime errors are sent to the event log and the**Batch Description** window using the following format:

```
"A run-time error 6 (Overflow) occurred in batch ErrorTest001 within the Index_Finish on
line 8 with node 278.
The priority of this batch will be set to 0 to prevent further problems."
```

The following example is a variation on the previous example. It includes a separate error handling function which is usable by the entire project:

```
Option Explicit

Public Function ErrorHandler(szStepName As String, oErr As Object, iErrLineNum As
Integer, oNode As Object)
1: Dim szErrMessage As String
2:
3: szErrMessage = "A run-time error " & oErr.Number & " (" & oErr.Description & ")
occurred in batch "
     & oNode.Tree.BatchName & " within the Index_Finish on line " & iErrLineNum & " with
node "
     & oNode.Tree.NodeID & ". The priority of this batch  be set to 0 to prevent further
problems."
7:
8: Call IALogNTMessage(&H1, szErrMessage)
9:
10: ' Set a message in the description box of the batch
11: Call IAValueSetAscii("$batch=" & oNode.Tree.BatchID & "/Description", szErrMessage)
12: ' Set the priority of the batch to zero
13: Call IAValueSetLong("$batch=" & oNode.Tree.BatchID & "/Priority", 0) 14:
End Function
```

Here is the new `Index_Finish`:

```
Private Sub Index_Finish(ByVal p As IASLib.IAS_RECORD_0)
1: Dim x As Integer
2:
3:
4: On Error GoTo Err_Handle
5:
6: x = 1
7:
8: x = (x - 1) / 0
9:
10: Exit Sub
11: Err_Handle:
13: ' Call our new error message handler
14: Call ErrorHandler("Index_Finish", Err, Erl(), p)
15:
End Sub
```

Note: Always use `Option Explicit` to prevent any typos from entering your code. Without `Option Explicit`, a typo could be turned into a new Variant VB variable, which could lead to unexpected behavior. Do not name your project `Err` or you would override the current `Information.Err` object that exists by default in VBA projects.

Special Notes on Handling Custom Export Module Errors

When most modules encounter an error while processing a task, the `Error` event handler for the step of the task is called. Many custom export modules, however, always return success status for a task, even when an error occurs. Modules like

ODBC Export and Documentum Advanced Export set an IA Value, generally `ExportResult`, to the appropriate error code. Then, they log an error message to the **Windows Event** log, and finish the task with a success status. The success status causes the module to call the `Finish` event handler for that step (where you would include code to deal with the error). The module calls the `Error` event handlers only if the task finishes with a failure status. A failure status is for example, loss of the connection to the Intelligent Capture Server.

Finishing the task with a success status, even though an error occurs, enables the custom export module to continue processing the remaining tasks in the batch. Most custom export module errors are due to one or several invalid index entries that the repository system cannot accept. In this case, you can reroute these pages to another module for correction while the custom export module continues processing the remaining tasks.

Special Notes on Handling RescanPlus and Copy Errors

By design, the Copy and RescanPlus modules both need to receive a task twice to process it. Each module returns an error code the first time it receives a task: RescanPlus returns status of `IA_ERR_CANCEL` to the Intelligent Capture Server and Copy returns a status of `IA_ERR_RETRY`. The task is requeued on the Intelligent Capture Server. The second time the module receives an error, it processes the task and either returns a success status (`IA_SUCCESS`) or an error code.

If you include an `Error` event handler in your *IPP* for RescanPlus or Copy, retrigger the task the first time that it is received. To retrigger the task, set <*RetriesLeft*> to a value greater than zero in your `Error` event handler for your *IPP*. The simplest way is to set the <*RetriesLeft*> value to itself.

8.7 Reference

The topics within this section contain reference information useful while using the application in setup or production.

8.7.1 Windows

The topics within this section provide descriptions of windows accessible from the application. The topics list the user interface element name and include a brief description of actions available from the window.

8.7.1.1 Add Step

To display the **Add Step** window, select **New Project** on the **File** menu and click the **Add** button in the **Define Steps** window.

Table 8-3: Add Step Window

Element	Description
Step	Type a **Step** name. For the maximum length allowed, see *OpenText Intelligent Capture - Operating Specifications (ECPCORE-RLI)*.
Module	Select the module name to be associated with the step.
Level	Select the level 0-7, used to trigger the module.
Departments	Type the name(s) of step-level departments to associate with the step. Departments are an optional classification within Intelligent Capture that enable an IPP to route tasks to specific steps. For the maximum length allowed, see *OpenText Intelligent Capture - Operating Specifications (ECPCORE-RLI)*.
Select MDFs	Displays the **Select MDFs** window.

8.7.1.2 Define Steps

To display the **Define Steps** window, select **New Project** from the **File** menu in Process Developer.

Table 8-4: Define Steps Window

Element	Description
Steps	Displays the step name, the module, the level used to trigger the module, and the departments defined in the step.
Add	Displays the **Add Step** window.
Remove	Removes the module step from the **Steps** list.
Modify	Displays the **Modify Step** window.
Options	Displays the **Options** window used to select auto or manual retriggering when the tree changes .
Select MDFs	Displays the **Select MDFs** window.

8.7.1.3 Install Process

To display the **Install Process** window, select **Install Process** from the **File** menu in Process Developer.

Table 8-5: Install Process Window

Element	Description
File name	Type the path and file name of the compiled process file that you want to install.
Browse	Displays the **Find Process (.iap) File** window which allows you to select the location of a compiled process file.
Process name	Type a name for the process. For the maximum length allowed, refer to the *Intelligent Capture Limits* topic in *Intelligent Capture Guide*. The process name may contain the following characters: • A-Z, a-z, 0-9, <space> • - _ ' ' ~ # ! @ $ % [ ] { } ( ) +
Install Process	Installs a compiled process file (*IAP*) onto an Intelligent Capture Server.

8.7.1.4 Modify Step

To display the **Modify Step** window, select **Modify Steps** from the **File** menu in Process Developer and double-click a step from the **Modify Steps** window. The **Modify Step** window displays.

Table 8-6: Modify Step Window

Element	Description
Step	Displays the name of the **Step**. Type a new name to change the step name and click **OK**. For the maximum length allowed, see *OpenText Intelligent Capture - Operating Specifications (ECPCORE-RLI)*.
Module	Select the module name to be associated with the step.
Level	Select the level 0-7, used to trigger the module.

Element	Description
Departments	Type the names of the departments to be associated with the step. Departments are an optional classification within Intelligent Capture that enable an *IPP* to route tasks to specific steps. For the maximum length allowed, see *OpenText Intelligent Capture - Operating Specifications (ECPCORE-RLI)*.
Select MDFs	Displays the **Select MDFs** window.

8.7.1.5 Modify Steps

To display the **Modify Steps** window, select **Modify Steps** from the **File** menu in Process Developer.

Table 8-7: Modify Steps Window

Element	Description
Steps:	Displays the step name, the module name, the level used to trigger the module, and the departments defined in the step.
Add	Displays the **Add Step** window.
Remove	Removes the module step from the steps list.
Modify	Displays the **Modify Step** window.
Options	Displays the **Options** window used to select auto or manual retriggering when the tree changes.
Select MDFs	Displays the **Select MDFs** window.

8.7.1.6 Options

Intelligent Capture uses automatic retriggering by default. You can disable automatic retriggering by selecting **Manual Retriggering** from the **Options** window in the **Define Steps** or **Modify Steps** windows.

Table 8-8: Options Window

Element	Description
Auto-retriggering (default)	Automatically retriggers tasks contained in a tree structure that is modified during processing.
Manual retriggering	When selected, tasks are not automatically retriggered in a tree structure is modified during processing.

8.7.1.7 Select MDFs

To display the **Select MDFs** window, select **Modify Steps** from the **File** menu in Process Developer, double-click a **Step** from the **Modify Steps** window. The **Modify Step** window displays, select **MDFs** to display the **Select MDFs** window.

Table 8-9: Select MDFs Window

Element	Description
Modules List	Adds the *MDF* File to an *IPP*. The Modules List: contains the following information: *MDF* File, Module(s), Directory, and Defined Types
Add MDF	Displays the **Open** window.
View	Displays the **View MDF** window which contains the Module Definition File.

8.7.1.8 Process Developer

This section lists the standard Microsoft Visual Basic for Applications commands and customized commands which are available when developing and debugging using Process Developer. All commands are listed under their menu heading.

8.7.1.8.1 File Menu

The **File** menu of the Process Developer window contains the following elements.

Table 8-10: File Menu

Element	Shortcut	Description
New Project	**CTRL+N**	Displays the **Define Steps** window, used to create an *IPP*.
Open Project	**CTRL+O**	Displays the **Open** window, used to open an existing *IPP*.
Close Project		Closes the active *IPP*.
Open Batch		Displays the **Open Batch** window to select a batch during debugging.
Close Batch		Closes an open batch.
Save	**CTRL+S**	Saves changes to the active *IPP*.

Element	Shortcut	Description
Save As		Displays the **Save As** window, used to save changes made to the active batch or *IPP*.
Import File	**CTRL+M**	Enables you to import a Visual Basic module or class into your *IPP*.
Export File	**CTRL+E**	Enables you to export a Visual Basic module or class from your *IPP*.
Remove		Deletes the selected Visual Basic module or class from your *IPP*.
Print	**CTRL+P**	Prints the highlighted selection, Visual Basic module, or *IPP*.
Modify Steps		Displays the **Modify Steps** window, enabling you to edit steps in the active *IPP*.
Make IAP		Compiles the *IPP* into an *IAP* file.
Install Process		Displays the **Install Process** window, enabling you to install a compiled *IAP* onto an Intelligent Capture Server.
Exit	**ALT+Q**	Closes Process Developer.

8.7.1.8.2 View Menu

The **View** menu of the Process Developer window contains the following elements.

Table 8-11: View Menu

Element	Shortcut	Description
Code	**F7**	Displays code for the item selected in the **Project Explorer** window.
Definition	**SHIFT+F2**	Displays the location in the **Code** window where the variable or procedure under the pointer is defined.
Last Position	**CTRL+SHIFT+F2**	Moves the insertion point to the beginning of the last line of edited code.

Element	Shortcut	Description
Object Browser	**F2**	Displays the **Object Browser** window, used to view the object libraries, the type libraries, classes, methods, properties, events, and constants used in code, as well as the modules, events, and procedures you defined for your *IPP*.
Immediate Window	**CTRL+G**	Displays the **Immediate** window, used to run statements one at a time.
Locals Window		Displays the **Locals** window, used to view the value of variables.
Watch Window		Displays the **Watch** window, used to view the values of variables and expressions.
Call Stack	**CTRL+L**	Displays the **Call Stack** window, used to view active events, procedures, and functions.
Project Explorer	**CTRL+R**	Displays the **Project Explorer** window, which displays your *IPP* and its components.
Properties Window	**F4**	Displays the **Properties** window, used to set and check the values of object properties.

8.7.1.8.3 Insert Menu

The **Insert** menu of the Process Developer window contains the following elements.

Table 8-12: Insert Menu

Element	Description
Procedure	Displays the **Add Procedure** window, used to insert a new event handler or procedure into your code.
Module	Inserts a new Visual Basic module into your *IPP*.
Class Module	Inserts a new Visual Basic class module into your *IPP*.

Element	Description
File	Displays the **Insert File** window, used to insert a text, Basic, or class file into your code.

8.7.1.8.4 Debug Menu

The **Debug** menu of the Process Developer window contains the following elements. These commands are disabled in Design mode.

Table 8-13: Debug Menu

Element	Shortcut	Description
Step Into	**F8**	Runs one statement of the *IPP*.
Step Over	**SHIFT+F8**	Runs the entire *IPP*.
Step Out	**CTRL+SHIFT+F8**	Runs all of the remaining statements in the *IPP*.
Run To Cursor	**CTRL+F8**	Runs all of the statements in the *IPP* up to the insertion point.
Add Watch		Displays the **Add Watch** window, used to add variables and expressions to the watch list.
Edit Watch	**CTRL + W**	Displays the **Edit Watch** box, used to edit variables and expressions in the watch list.
Quick Watch	**SHIFT+F9**	Displays the **Quick Watch** box, which displays the values of selected variables and expressions in the **Code** window.
Toggle Breakpoint	**F9**	Creates and deletes breakpoints.
Clear All Breakpoints	**CTRL+SHIFT+F9**	Deletes all breakpoints in the selected code.
Set Next Statement	**CTRL+F9**	Continues running code at the insertion point.
Show Next Statement		Highlights the next statement to be run.

8.7.1.8.5 Run Menu

The **Run** menu of the Process Developer window contains the following elements. These commands are disabled in Design mode.

Table 8-14: Run Menu

Element	Shortcut	Description
Continue	**F5**	Resumes running the procedure in which the cursor is placed.
Break	**CTRL+BREAK**	Stops running the selected event handler or procedure, placing the code in Break mode.
Reset		Resets all Visual Basic module-level variables and clears the Call Stack.

8.7.1.8.6 Tools Menu

The **Tools** menu of the Process Developer window contains the following elements. These commands are disabled in Design mode.

Table 8-15: Tools Menu

Element	Description
Connect to Server	Connects to an Intelligent Capture Server.
Disconnect from Server	Disconnects from the Intelligent Capture Server to which you are currently connected.
Options	Displays the **Options** window, used to customize the Visual Basic Editor.
Properties	Displays the **Project Properties** window, from which you specify the properties of your *IPP*.

8.7.1.8.7 Windows Menu

The **Windows** menu of the Process Developer window contains the following elements.

Table 8-16: Windows Menu

Elements	Description
Split	Splits/removes splits from the **Code** window.
Tile Horizontally	Horizontally tiles all non-minimized **Code** windows.
Tile Vertically	Vertically tiles all non-minimized **Code** windows.
Cascade	Cascades all non-minimized **Code** windows into an overlapping arrangement.
Arrange Icons	Arranges all minimized **Code** windows into rows at the bottom of the main window.

8.7.1.8.8 Help Menu

The **Help** menu of the Process Developer window contains the following elements.

Table 8-17: Help Menu

Element	Shortcut	Description
Microsoft documentation	**F1**	Displays the Microsoft *Visual Basic Reference*.
Contents		Displays the *Process Developer Guide*.
About Process Developer		Displays name and version information for Process Developer.
About Microsoft Visual Basic		Displays name and version information for Microsoft Visual Basic.

8.7.2 Programming Reference

This section describes programming reference information for *MDF*s and *IPP*s.

8.7.2.1 MDF Reference

This section describes the elements included in an *MDF*.

8.7.2.1.1 Triggers Defined in Default MDFs

This section lists Intelligent Capture modules and utilities with the names of their corresponding *MDF*s. It also lists the trigger levels and the trigger IA Values for each module. Use trigger values in an *IPP* to route tasks to the module during processing. Those modules and utilities that are not intended for use within processes, such as the Intelligent Capture Administrator, are not listed here.

Table 8-18: Intelligent Capture Modules

Module	MDF Name	Trigger Level(s)	Trigger IA Value(s)
.NET Code Module	`code.mdf`	0–7	*<Ready>*
ApplicationXtender Export	`exax.mdf`	0–7	*<Ready>*
Archive Export	`exsapal.mdf`	0–7	*<Ready>*
Completion	`cpdsktop.mdf`  **Note:** When adding this MDF file, ensure that the `SharedObjects.mdf` file is also added.	1–7	*<Ready>*
Classification	`dpclssf.mdf`	0–7	*<InputImage>*
Identification	`dpclssfe.mdf`	0–7	*<InputImage>*
Collector	`dpcollec.mdf`	0–7	*<InputImage>*
Copy	`iacopy.mdf`	7	*<Ready>*
Documentum Advanced Export	`iaexdm.mdf`	0-7	*<InputImage>*
Extraction	`cpextrac.mdf`	1–7	*<Image>*
FileNet Content Manager Export	`exfncm.mdf`	0-7	*<Ready>*
FileNet Panagon IS/CS Export	`iaxfnet2.mdf`	0-7	*<InputImage>* *<Level<n>_InputFile>*

Module	MDF Name	Trigger Level(s)	Trigger IA Value(s)
Global 360 Export (formerly known as eiStream WMS Export)	iaexwnt.mdf	0-7	*<InputImage>*
Export for IBM Content Manager	exicm.mdf	0 0-7	*<InputImage>* *<Level<n>_InputFile1>* *<Level<n>_InputFile2>* *<Ready>*
Export for SAP Archive and AP Connect	excssap.mdf	0	*<InputImage>*
Image Converter	imgconv.mdf	0-7	*<InputFile>*
Image Processor	cpimgpro.mdf	0	*<InputFile>*
MS SharePoint Export	exshrpt2.mdf	0–7	N/A
Multi	iamulti.mdf	0	*<Ready>*
NuanceOCR	ssocr.mdf	0 0-7	*<Level<n>_InputImage>* *<Ready>*
ODBC Export	iaxodbc2.mdf	0-7	*<InputImage>*
Export for OpenText Content Server	exll2.mdf	0 0-7	*<InputImage>* *<Level<n>_InputFile1>* *<Level<n>_InputFile2>*
RescanPlus	rescanplus.mdf	0–7	*<InputImage>* *<InputImage2>* *<InputImage3>* *<NeedsRescan>*
ScanPlus	scanplus.mdf	N/A	N/A
Standard Export	cpexport.mdf	0–7	N/A
Timer	iatimer.mdf	N/A	N/A
WS Input	wsinput.mdf	0	*<InputImage>*
WS Output	wsoutput.mdf	0	*<InputImage>*

8.7.2.1.2 MDF Value Data Types

The following data types are allowed in the Module section:

- Integer: 2-byte integer.
- Long: 4-byte long integer.
- Boolean: 1-byte Boolean.
- File: Input or output file reference.
- String: Variable-length string.
- String*n: Fixed-length string.
- Single: 4-byte single-precision floating point.
- Double: 8-byte double-precision floating point.
- Currency: 8-byte scaled integer.
- Date: 8-byte date.
- Object: Data type that contains a set of member variables.
- IAVariant: Data type that can represent different objects.

8.7.2.1.3 MDF Value Attributes

In addition to specifying the data type of each *MDF* value, each data type can have several attributes. Attributes immediately follow the data type. Separate the attributes with commas. These attributes are:

Table 8-19: MDF Value Attributes

MDF Value Attributes	Description
`Input`	An input variable or file. This is the default and can be omitted.
`Output`	An output variable or file.
`Prime`	An input variable that is automatically sent to the module along with the task (default for input variables).
`Noprime`	A variable that is not automatically sent to the module.
`Prefetch`	An input file that is automatically sent to the module along with the task (default for input files).
`Noprefetch`	A file that is not prefetched automatically.

MDF Value Attributes	Description
`Trigger`	A value that is a trigger. Only the types Integer, Long, and File can be triggers. Any value of type File, declared as an Input value, is automatically a trigger, unless it is declared as Notrigger.
`Notrigger`	A variable that is not a trigger.
`Format`	Formatting for a variable, including justification (left or right) and padding characters. This attribute is only valid with `String*`*<n>* values.
`Const`	A variable that is a constant. Only members of an object can be declared a constant. A variable that is declared as a constant is read-only and requires an initial value within the *MDF* (similar to a default value). Attempting to set a constant *MDF* Value results in a run-time error. IA Values that use the `Const` attribute are saved in the batch file only once per Object type to save space.

8.7.2.2 IPP Reference

This section describes the elements included in an *IPP*.

8.7.2.2.1 Visual Basic Reference

Although most Visual Basic code is supported, the "Unsupported Visual Basic Features" on page 550 section contains a list of limitations when developing an *IPP*.

8.7.2.2.2 Unsupported Visual Basic Features

In addition to the Intelligent Capture Server methods, events, and properties described in this section, you can use most of the standard functions provided with VBA in your *IPP*, with the following exceptions:

Table 8-20: Unsupported Visual Basic Features

Element	Description
Screen input and output functions	Screen input and output functions such as `MsgBox` or `InputBox` is not supported. If you include these in your process, then the resulting user interface only displays on the Intelligent Capture Server console, not on the client machine. If the option requires user input, production is halted and the operator must have direct access to the Intelligent Capture Server console to resume processing.
ODBC calls	`ODBC` calls such as `SQLGetSchema` or `SQLExecQuery` are not supported. Instead, use ODBC Export to route data to an `ODBC` data source. The *ODBC Export Guide* contains more information.
Network and *CPU*-intensive *VBA* operations	All *VBA* execution is single-threaded within an Intelligent Capture Server instance and runs inside the Intelligent Capture Server process. Thus, you must avoid possible long-running and high-load *VBA* operations, such as: • Access to external back-end systems, such as Documentum, SAP, SQL Server, and others, • Calls to an external system using a Web service, • Local and network file access and other network operations, • Calls to any third-party libraries or external processes, • Long-running computations, complex data analysis, and other *CPU* and memory-intensive operations. Instead, put this type of code in a script within a client module or use a dedicated code module.

Element	Description
Global and Static variables	Using `Global` and `Static` variables in an *IPP* is not supported. Their life span is indeterminate and they generally cause unpredictable effects. These variables sometimes retain their value between events, so they can appear persistent. However, whenever the VB project is unloaded, their values are lost. The *IPP* writer cannot know when the VB project is unloaded and this may change in future versions of the Intelligent Capture Server. **Note:** A variable is "persistent" if it retains its value when the Intelligent Capture Server syncs, the batch in which it is used is unloaded, or if the Intelligent Capture Server is shut down and restarted. IA Values are persistent, but variables declared within VB are not. This is true even if you use the Global and Static keywords provided by the VBA Language.
Type libraries	Use of any type libraries that are not installed with the product is not supported.
Direct *DLL* calls	Use of direct *DLL* calls is not supported.
Setting trigger variables	Certain ways of setting trigger variables is not supported. The "Working with MDFs" on page 482 and "Referencing Triggers" on page 528 sections explain these limitations.
Adding a BAS file to the *IPP*.	Adding BAS files as a class module is not supported. BAS files can be added as a new *VBA* module, however.

8.7.2.2.3 Intelligent Capture VBA Language Additions

Constants

Error Codes

The Common_Constants module includes declarations of the following error constants:

Table 8-21: Error Constants

Element	Error No.
`IA_SUCCESS`	`0`
`IA_ERR_CANCEL`	`-4526`
`IA_ERR_NORETRY`	`-6112`
`IA_ERR_RETRYSOME`	`-6113`
`IA_ERR_RETRY`	`-6114`

Note: A complete list of client module error and log codes is listed in *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

Windows Event Types

The Common_constants module includes declarations of `Ialogntmessage`:

Table 8-22: Windows Event Types

Element	Description
`Ialogntmessage(`*`itype`*`, `*`strmessage`*`)`	`Ialogntmessage` writes messages to the Windows Application event log on the computer hosting the Intelligent Capture Server.

Note: The "IALogNTMessage Method" on page 581 section contains more about `Ialogntmessage` and its associated constants.

Event Handlers

This section describes the event handlers that the process can use.

Note: Use `Option Explicit` in all event handlers to prevent typographical errors. Without `Option Explicit`, a typographical error could be turned into a new Variant VB variable, which could lead to unexpected behavior. You can automate the addition of `Option Explicit` to all your steps by enabling the **Require Variable Declaration** option. This option only applies to new steps that you add to an *IPP* (or all steps in a new *IPP*. Before creating an *IPP*, open one of the sample *IPP*s to enable the **Options** menu command in the **Tools** menu. In the **Editor** tab, select the **Require Variable Declaration** checkbox. All new *IPP*, include `Option Explicit` in all steps.

Task-driven Events

For each module step in your process, there are five event handlers that can be defined at the task level. Three of these event handlers are task-driven: `Error`, `Finish`, and `Prepare`. The three task-driven events and when they are called are:

- `Error`: Called when a module fails to process a task.
- `Finish`: Called when a module has finished processing a task.
- `Prepare`: Called when a module has a task and is ready to process it.

Module-requested Events

For each module step in your process, there are five event handlers that can be defined at the task level. Modules can call two of these event handlers, regardless of task status: `Notify` and `Retrigger`.

- `Notify`: Called by a module to inform the *IPP* that data has been changed for this task-level node.
- `Retrigger`: Called by a module to request retriggering of this task.

Tree Manipulation Events

There is a special step object, called `Tree`, which contains event handlers for tree manipulation events.

At each level in the tree (excluding level 7), there are four tree manipulation event handlers that can be defined. The four tree manipulation event handlers are:

- `PostNodeAdd<n>`: Called immediately after a node is added to the tree.
- `PostNodeMove<n>`: Called immediately after one or more nodes are moved in the tree.
- `PreNodeDelete<n>`: Called immediately before a node is deleted from the tree.
- `PreNodeMove<n>`: Called immediately before one or more nodes are moved in the tree.

Process Events

There is a special step object available in Process Developer, called “Process”, which contains two event handlers:

- `Install`: Called when the process is first installed (immediately after `PreInstall`) and it is editable by the *IPP* developer.
- `PreInstall`: Called when the Process Developer module first install and generates the process automatically.

When a process is first installed to the Intelligent Capture Server, these two events are called. These events can be used to set default values.

Batch Events

There is a special step object available in Process Developer called “Batch” that contains two event handlers.

When a batch is created or deleted, one of the following events is called:

- `Create`: Called when the batch is created.
- `Delete`: Called immediately before a batch is deleted.

`IIAS_Info Object`

Many of the event handlers use `IIAS_Info object` to tell the *IPP* who caused the event. The events which include this information are the `Notify` and `Retrigger` events, the tree manipulation events, and the batch and process events.

Table 8-23: `IIAS_Info Object`

Property	Description
`Step`	A constant of type String, containing the step name of the calling module. **Note:** The step name is only populated if the calling module has active tasks. The step name is associated with the task. If the calling module has no active tasks, or if it has active tasks from multiple steps (pre-fetched tasks, for step), then the `Step` property may be blank
`Module`	A constant of type String containing the name of the module that sent this event.
`User`	A constant of type String containing the user name that the calling module used to log into the Intelligent Capture Server.

Create Event

The `Create` event handler is called after a batch is created, enabling the *IPP* to set default values. The *IPP* can also override the default values set in the `Process_PreInstall` and `Process_Install` routines.

Syntax

```
Private Sub Batch_Create(ByVal <pRoot> As IASLib.IAS_RECORD_7, ByVal <p6> As
IASLib.IAS_RECORD_6,
ByVal <p1> As IASLib.IAS_RECORD_1, ByVal <p> As IASLib.IAS_RECORD_0, ByVal <pInfo> As
IASLib.IAS_INFO)
```

The `Create` event handler syntax has the following parts:

Table 8-24: `Create` Event Handler Syntax

Part	Description

`pRoot`
 	
`p<n>`
 	
`p`
	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch).
	
	The level 0 through level 6 nodes do not represent actual nodes in the tree, but are merely placeholders for default values. All new nodes added to the tree inherit default values from these placeholder nodes.
	
	The level 7 node already exists (the batch has been created), so the any level 7 IA Values are stored on the existing level 7 node.

`pInfo`	An object of type IIAS_Info containing information about who created this batch.

Comments

The `Create` event handler provides a mechanism for an *IPP* to set default values for IA Values when a batch is created. A node object for each level is passed to this event handler, allowing you to set default values for any level node. By the time the `Create` event handler is called, the batch has already been created. Therefore, the *`pRoot`* node is an actual node in the tree (the only node in the tree, in fact).

However, the level 0 through level 6 nodes which are passed to this event handler are not actual nodes in the tree. Do not use any Tree navigation properties or methods with these nodes. They are placeholder nodes for storing default IA Values. Any new nodes created in this batch inherit their nodal values from these placeholder nodes.

The `Create` event handler is called just after the batch is created. Therefore, you can set default values which depend on data that is not available until the batch is created (such as the date and time). However, using the `Process_Install` event handler is slightly more efficient, because it is only called (when the process is installed).

The *`pInfo`* object contains the name of the module which created the batch, as well as the user who was logged in to the module. However, the step name is populated, because steps are associated with tasks. A step is not associated with the process of creating batch.

If necessary, your *IPP* could attempt to guess the step, based on the module. For example, if `pInfo.Module` is ScanPlus, then your *IPP* could assume that the step is ScanPlus, or whatever the ScanPlus step is called in your *IPP*.

Delete Event

The `Delete` event handler is called just before a batch is deleted. Therefore, the *IPP* can log values to the s Event Log using the "IALogNTMessage Method" on page 581 or the `debug.out` file using "`IALogDebug` Method" on page 580.

Syntax

```
Private Sub Batch_Delete(ByVal <pRoot> As IASLib.IAS_RECORD_7, ByVal <pInfo>As
IASLib.IAS_INFO)
```

The `Delete` event handler syntax has the following parts:

Table 8-25: `Delete` Event Handler Syntax

Part	Description
`pRoot`	An object of type `IASLib.IAS_RECORD_7` containing the record structure of level 7, the batch level. All other nodes in the tree can be reached by using the Tree navigation properties.
`pInfo`	An object of type IIAS_Info containing information about who is deleting this batch.

Comments

You can use the `Delete` event handler to save information about a batch before it is deleted (for an audit trail, for example). The *IPP* cannot prevent the batch from being deleted. Therefore, any information you want to save must be logged to the Windows Event Log or the `debug.out` file.

Error Event

An `Error` event handler is called when a module fails to process a task. The module returns the error. If the module disconnects or crashes, the Intelligent Capture Server returns the error.

Syntax

```
Private Sub <Stepname>_Error(ByVal <p> As IAS_RECORD_<n>, ByVal <Code> As Integer)
```

The `Error` event handler syntax has the following parts:

Table 8-26: `Error` Event Handler Syntax

Part	Description
`p`	An object of type `IASLib.IAS_RECORD_`*<n>* containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch). <*n*> is the level that the module is set to trigger at in the Define Steps or Modify Steps window.
`StepName`	The name of the module step as defined in the **Define Steps** window.
`Code`	A constant of type Integer identifying the error handling mode, defined in a non-class module.

Comments

Create an `Error` event handler for each module whose errors you want to handle. Some modules are more prone to task-related errors than others. For example, the OCR and Image Processor modules can receive images they cannot handle. By using the `Error` event handler, a process can trap, resubmit, and reroute page nodes that cause errors.

Example

This example checks the error code of the step named OCR, and then changes the <*RetriesLeft*> value accordingly. If <*RetriesLeft*> decrements to 0, then the task status becomes "Error".

```
Option Explicit

Private Sub OCR_Error(ByVal p As IASLib.IAS_RECORD_0, ByVal Code As Integer)
    Select Case Code
        Case IA_ERR_RETRYSOME
            p.OCR.RetriesLeft = p.OCR.RetriesLeft - 1
        Case IA_ERR_NORETRY
            p.OCR.RetriesLeft = 0
        Case IA_ERR_RETRY
            ' Setting this IA Value to itself
            ' causes the InputAccel Server to
            ' reevaluate the Status of the task.
            p.OCR.RetriesLeft = p.OCR.RetriesLeft
        Case IA_ERR_CANCEL
            p.OCR.RetriesLeft = p.OCR.RetriesLeft
        Case Else
            p.OCR.RetriesLeft = p.OCR.RetriesLeft - 1
    End Select

    ' If RetriesLeft is 0, the task becomes
    ' "Error". This can hold up the batch, so
    ' send the page to RescanPlus.
    If OCR.RetriesLeft = 0 then
        'Go to Rescan
        p.RescanPlus.InputImage = p.OCRFull.InputImage
        p.RescanPlus.RescanReason = "Didn't go."
        p.RescanPlus.NeedsRescan = 3
```

```
    End if
End Sub
```

Finish Event

A `Finish` event handler is called when a module has finished processing a task.

Syntax

```
Private Sub <Stepname>_Finish(ByVal <p> As IAS_RECORD_<n>)
```

The `Finish` event handler syntax has the following parts:

Table 8-27: `Finish` Event Handler Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_<*n*> containing the record structure of Level <*n*>, where <*n*> is an integer from 0 (Page) through 7 (Batch). <*n*> is the level at which the module is set to trigger in the **Define Steps** or **Modify Steps** window.
`StepName`	The name of the module step as defined in the **Define Steps** window.

Comments

You can use `Finish` event handlers to "connect" one module to the next. Specifying that the output files from one step must be used as the input files for the next step. With many custom export modules, `Finish` event handlers can also include error handling code. The "Inserting an Event Handler" on page 515 section contains information about creating the "shell" of a `Finish` event handler.

Example

This example routes the `OutputImage` of the step named "Scan" to the <*InputImage*> of the step named "ImageExp".

```
Option Explicit
Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
        p.ImageExp.InputImage = p.Scan.OutputImage
End Sub
```

Install Event

The `Install` event handler is called after a process is installed to the Intelligent Capture Server, enabling the *IPP* to set default values. The `Install` event is called after the `PreInstall` event, allowing your process to override the defaults set in the `PreInstall` event.

The syntax is as follows:

```
Private Sub Process_Install(ByVal <pRoot> As IASLib.IAS_RECORD_7,
ByVal <p6> As IASLib.IAS_RECORD_6, ...ByVal <p1> As IASLib.IAS_RECORD_1,
ByVal <p> As IASLib.IAS_RECORD_0, ByVal <info> As IASLib.IAS_INFO)
```

The `Install` event handler syntax has the following parts:

Table 8-28: `Install` Event Handler Syntax

Part	Description

`pRoot`
 	
`p`*<n>*
 	
`p`
	An object of type *IAS*Lib.IAS_RECORD_*<n>* containing the record structure of Level *<n>*, where *<n>* is an integer from 0 (page) through 7 (batch).
	
	These nodes do not represent actual nodes in the tree, but are merely placeholders for default values. All new nodes added to the tree inherits default values from these placeholder nodes.

`pInfo`	An object of type `IIAS_Info` containing information about who installed this process.

Comments

The `Install` event handler provides a mechanism for an *IPP* to set default values for IA Values, *MDF* values, dynamic values, and non-nodal values). A node object for each level is passed to this event handler, enabling you to set default values for any level node. The nodes passed to this event handler are not actual nodes in the tree. However, any new nodes created in batches based on this process inherits their nodal values from these placeholder nodes.

Note: When the `Process_Install` event handler is called, there is not an actual tree structure. The tree is not created until a batch is created. Therefore, do not attempt to use any tree navigation properties or methods (such as `pRoot.Tree.NumChildren(0)`). Doing so causes a runtime error in your *IPP*.

The `PreInstall` event is called before `Install`. The *MDF* parser uses `PreInstall` to set default values from the *MDF*s. You can use `Install` in the *IPP*to override those default values or to set other values.

The `pInfo` object contains the name of the module which created the batch, as well as the user who was logged in to the module. However, the step name is populated because steps are associated with tasks. A step is not associated with installing a process.

Example

This example sets the default value for a level 1 *MDF* Value for the “Index” step. It also creates a dynamic value with a default for the root node.

```
Option Explicit

Private Sub Process_Install(ByVal pRoot As IASLib.IAS_RECORD_7,
ByVal p6 As IASLib.IAS_RECORD_6, ByVal p5 As IASLib.IAS_RECORD_5,
ByVal p4 As IASLib.IAS_RECORD_4, ByVal p3 As IASLib.IAS_RECORD_3,
ByVal p2 As IASLib.IAS_RECORD_2, ByVal p1 As IASLib.IAS_RECORD_1,
ByVal p As IASLib.IAS_RECORD_0, ByVal pInfo As IIAS_Info)

    p1.Index.Level1_Index0 = "Not reviewed"
    pRoot.Scan("Process_Install_Date") = Now
End Sub
```

Notify Event

A module can use the `Notify` event handler to perform any necessary function for a particular step on a task-level node. The module that calls the function (through the client *API*, which tells the server to run the function) provides the ***`Reason`*** string. The `Notify` event can be called for any task-level node, regardless of whether that node is an active task or not.

Note: To use the `Notify` event handler, you must have a module that has been designed to call the `IABatchCallNotify` client *API* function. The aim is to inform the *IPP* that data has been manipulated on a non-task node. The `Finish` event handler would not be called for the non-task node). The `Notify` event handler is a function, not a subroutine. Therefore, it can return a value to the Intelligent Capture Server.

Syntax

```
Private Function <Stepname>_Notify(ByVal <p> As IAS_RECORD_<n>,
ByVal <Reason> As String, ByVal <pInfo> As IIAS_Info) As String
```

The `Notify` event handler syntax has the following parts:

Table 8-29: `Notify` Event Handler Syntax

Part	Description
`p`	An object of type `IASLib.IAS_RECORD_`*<n>* containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch). <*n*> is the level at which the module is set to trigger in the **Define Steps** or **Modify Steps** window.
`StepName`	The name of the module step as defined in the **Define Steps** window.
`Reason`	A constant of type String containing a value sent by the module to the *IPP*.
`pInfo`	An object of type `IIAS_Info` containing information about who called the `Notify` event handler.

The return value of this function is a string that is passed back to the module.

Comments

The `Notify` event handler provides a mechanism for modules to send information to the *IPP* without having to finish a task. The following characteristics of the `Notify` event handler set it apart from the `Error`, `Finish`, and `Prepare` event handlers. A module can call the `Notify` event handler:

- Without having received a task.
- For any task-level node in the tree. Nodes can be task nodes processed by another module, nodes that are not currently being processed, or even nodes not ready for processing.
- For any step in the *IPP*. The step can be a step of the calling module or not.

Because of these differences, the `Notify` event handler gives a module greater flexibility in controlling the process flow than the `Finish` event handler. However, specifically design the module to call the `Notify` event handler.

PostNodeAddn Event

The `PostNodeAdd<n>` event handler is called after a node is added to the tree. Nodes include page nodes (scanned or imported), as well as higher level nodes. A `PostNodeAdd<n>` event handler can be defined for each level in the tree, except level 7.

Note: Inserting a level 1 or higher node into the tree splits a document and calls multiple event handlers in a certain order. The "Event Order when Splitting a Document" on page 516 section contains more information.

Syntax

```
Private
Sub Tree_PostNodeAdd<n>(ByVal p As IAS_RECORD_<n>, ByVal pInfo As IIAS_Info)
```

The `PostNodeAdd<n>` event handler syntax has the following parts:

Table 8-30: `PostNodeAdd<n>` Event Handler Syntax

Part	Description
`p`	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <n>, where <n> is an integer from 0 (Page) through 6. You can have up to 7 `PostNodeAdd<n>` event handlers, one for each level from 0 to 6.
`pInfo`	An object of type `IIAS_Info` containing information about who made this modification to the tree.

Comments

The `PostNodeAdd<n>` event handler provides a mechanism for an *IPP* to perform any necessary tasks when a node is added to the tree. A PostNodeAdd<*n*> can be created for each level in the tree, from 0 to 6. Although there is no `PostNodeAdd<n>` event for level 7, there are events for batch creation and process installation.

The `PostNodeAdd<n>` event can be limited in usefulness, because of the way the tree is manipulated when a node is split.

PostNodeMoven Event

The `PostNodeMove<n>` event handler is called just after one or more nodes are moved in the tree. A `PostNodeMove<n>` event handler can be defined for each level in the tree, except level 7.

Note: Inserting a level 1 or higher node into the tree splits a document and calls multiple event handlers in a certain order. The "Event Order when Splitting a Document" on page 516 section contains more information.

Syntax

```
Private Sub Tree_PostNodeMove<n>(ByVal pDest As IASLib.IAS_RECORD_<m>,
ByVal pSrc As IASLib.IAS_RECORD_<m>, ByVal pNodes As IASLib.PageCollection, ByVal
pInfo As IIAS_Info)
```

The `PostNodeMove<n>` event handler syntax has the following parts:

Table 8-31: `PostNodeMove<n>` Event Handler Syntax

Part	Description
pDest	An object of type `IASLib.IAS_RECORD_<m>` (<*m*>=<*n*>+1) containing the record structure of level <*m*>, where <*m*> is an integer from 1 through 7. *pDest* is the new parent node for the node(s) that are being moved.
pSrc	An object of type `IASLib.IAS_RECORD_<m>` (<*m*>=<*n*>+1) containing the record structure of level <*m*>, where <*m*> is an integer from 1 through 7. *pSrc* is the old parent node for the node(s) that are being moved.
pNodes	A collection of records of type `IASLib.IAS_RECORD_<n>` containing the record structure of Level <*n*>, where <*n*> is an integer from 0 (Page) through 6. You can have up to 7 `PostNodeMove<n>` event handlers, one for each level from 0 to 6.

Part	Description
pInfo	An object of type IIAS_Info containing information about who made this modification to the tree.

Comments

The PostNodeMove<*n*> event handler provides a mechanism for an *IPP* to perform any necessary tasks after one or more nodes are moved in the tree. A PostNodeAdd<*n*> can be created for each level in the tree, from 0 to 6.

The PostNodeMove<*n*> event handler is called after a collection of nodes are moved. The PreNodeMove<*n*> event handler is called before moving the collection.

When a document node is split, then several events are called in a certain order. A document node is split by inserting a level 1 or higher node between two page nodes in the same document. The PostNodeMove<*n*> event is more useful in practice than the PostNodeAdd<*n*> event that is generated

PreInstall Event

The PreInstall event handler is called after a process is installed to the Intelligent Capture Server, enabling the *IPP* to set default values. The PreInstall event handler is automatically generated in Process Developer when the *MDF*s are parsed. The Install event is called after the PreInstall event, enabling your process to override the defaults set in the PreInstall event.

Note: Do not modify the code that is automatically generated for the PreInstall event handler. Every time you recompile your *IPP*, the *MDF*s are reparsed, generating a fresh PreInstall event handler. Any changes you have made are lost. Use the "Install Event" on page 559 to change the default values set in PreInstall.

Syntax

```
Private Sub Process_PreInstall(ByVal <pRoot> As IASLib.IAS_RECORD_7,
ByVal <p6> As IASLib.IAS_RECORD_6, ...ByVal <p1> As IASLib.IAS_RECORD_1,
ByVal <p> As IASLib.IAS_RECORD_0, ByVal <info> As IASLib.IAS_INFO)
```

The PreInstall event handler syntax has the following parts:

Table 8-32: `PreInstall` Event Handler Syntax

Part	Description

`pRoot`
 	
`p`*<n>*
 	
`p`
	An object of type `IASLib.IAS_RECORD_`*<n>* containing the record structure of Level *<n>*, where *<n>* is an integer from 0 (page) through 7 (batch).
	
	These nodes do not represent actual nodes in the tree, but are merely placeholders for default values. All new nodes added to the tree inherit default values from these placeholder nodes.

`pInfo`	An object of type `IIAS_Info` containing information about who installed this process..

Comments

The `PreInstall` event handler provides a mechanism for an *IPP* to set default values for IA Values, based on default values in the *MDF*s. The nodes which are passed to this event handler are not actual nodes in the tree; they are used for storing default values for new nodes.

The `PreInstall` event is called before `Install`. The *MDF* parser uses `PreInstall` to set default values from the *MDF*s. The *IPP* developer can use `Install` to override those default values or to set other values.

Example

This example uses a custom *MDF* with a couple of default values. The *MDF* is given first; it includes a module named `Custom`. The *IPP* in this example includes a step of `Custom` called `CustVals`. Following the sample *MDF* is the `Process_PreInstall` event handler that Process Developer generates automatically.

```
Module Custom(1)
    EmployeeID as Long
    State as String*2, Output = "CA"
    NumDependents as Integer, Output = 2
End Module

Option Explicit

Private Sub Process_PreInstall(ByVal pRoot As IASLib.IAS_RECORD_7,
ByVal p6 As IASLib.IAS_RECORD_6, ByVal p5 As IASLib.IAS_RECORD_5,
ByVal p4 As IASLib.IAS_RECORD_4, ByVal p3 As IASLib.IAS_RECORD_3,
ByVal p2 As IASLib.IAS_RECORD_2, ByVal p1 As IASLib.IAS_RECORD_1,
ByVal p As IASLib.IAS_RECORD_0, ByVal pInfo As IIAS_Info)

   THE FOLLOWING CODE IS AUTOMATICALLY GENERATED
   FROM THE MDF AND SHOULD NOT BE MODIFIED.
   USER INITIALIZATION MUST INSTEAD BE ADDED
   IN THE "Process_Install" EVENT HANDLER.

    p1.CustVals.State = "CA"
```

```
    p1.CustVals.NumDependents = 2
End Sub
```

PreNodeDeleten Event

The `PreNodeDelete<n>` event handler is called just before one or more nodes are deleted from the tree. A `PreNodeDelete<n>` event handler can be defined for each level in the tree, except level 7.

Syntax

```
Private Sub Tree_PreNodeDelete<n>(ByVal <pNodes> As IASLib.PageCollection, ByVal <pInfo>
As IIAS_Info)
```

The `PreNodeDelete<n>` event handler syntax has the following parts:

Table 8-33: `PreNodeDelete<n>` Event Handler Syntax

Part	Description
`pNodes`	A collection of records of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <n>, where <n> is an integer from 0 (Page) through 6. You can have up to 7 `PreNodeDelete<n>` event handlers, one for each level from 0 to 6.
`pInfo`	An object of type `IIAS_Info` containing information about who made this modification to the tree.

Comments

The `PreNodeDelete<n>` event handler provides a mechanism for an *IPP* to perform any necessary tasks before one or more nodes are deleted from the tree. A `PreNodeDelete<n>` can be created for each level in the tree, from 0 to 6. Although there is no `PreNodeDeleten` event for level 7, there is an event for batch deletion.

Example

This example uses the `PreNodeDelete2` event handler to detect when a user deletes a level 2 node. It records the node IDs of the deleted nodes and the user who deleted the nodes. This information is saved to a dynamic value on the parent node of the deleted nodes.

```
Option Explicit

Private Sub Tree_PreNodeDelete2(ByVal pNodes As IASLib.PageCollection, ByVal pInfo As
IIAS_Info)
    Dim p2 As IASLib.IAS_RECORD_2

    For Each p2 In pNodes
        p2.Tree.Parent.Index("NodeHistory_3") = p2.Tree.Parent.Index("NodeHistory_3") &
" : Node
        " & p2.Tree.NodeID & " deleted by " & pInfo.User
```

```
    Next
End Sub
```

PreNodeMoven Event

The `PreNodeMove<n>` event handler is called just before one or more nodes are moved in the tree. A `PreNodeMove<n>` event handler can be defined for each level in the tree, except level 7.

Note: Inserting a level 1 or higher node into the tree splits a document and calls multiple event handlers in a certain order. The "Event Order when Splitting a Document" on page 516 section contains more information.

Syntax

```
Private Sub Tree_PreNodeMove<n>(ByVal <pDest> As IASLib.IAS_RECORD_<m>,
ByVal <pSrc> As IASLib.IAS_RECORD_<m>, ByVal <pNodes As IASLib.PageCollection>,
ByVal <pInfo> As IIAS_Info)
```

The `PostNodeMove<n>` event handler syntax has the following parts:

Table 8-34: `PreNodeMoven` Event Handler Syntax

Part	Description
`pDest`	An object of type `IASLib.IAS_RECORD_<m>` (<m>=<n>+1) containing the record structure of level <m>, where <m> is an integer from 1 through 7. <pDest> is the new parent node for the node(s) that are being moved (such as, the parent node that the nodes in `<pNodes>` are moving to).
`pSrc`	An object of type `IASLib.IAS_RECORD_<m>` (<m>=<n>+1) containing the record structure of level <m>, where <m> is an integer from 1 through 7. <pSrc> is the old parent node for the node(s) that are being moved (such as, the parent node from which the nodes in `<pNodes>` are moving).
`pNodes`	A collection of records of type `IASLib.IAS_RECORD_<n>` containing the record structure of Level <n>, where <n> is an integer from 0 (Page) through 6. You can have up to 7 `PostNodeMove<n>` event handlers, one for each level from 0 to 6.
`pInfo`	An object of type `IIAS_Info` containing information about who made this modification to the tree. The "`IIAS_Info` Object" on page 555 section contains more information.

Comments

The `PreNodeMove<n>` event handler provides a mechanism for an *IPP* to perform any necessary tasks before one or more nodes are moved in the tree. A `PreNodeAdd<n>` can be created for each level in the tree, from 0 to 6.

The `PreNodeMove<n>` event handler is called before a collection of nodes are moved. The `PostNodeMove<n>` event handler is called just after moving the collection.

When a document node is split, then several events are called in a certain order. A document node is split by inserting a level 1 or higher node between two page nodes in the same document.

Prepare Event

A `Prepare` event handler is called when a module has a task and is ready to process it.

Syntax

```
Private Sub <Stepname>_Prepare(ByVal <p> As IAS_RECORD_<n>)
```

The `Prepare` event handler syntax has the following parts:

Table 8-35: `Prepare` Event Handler Syntax

Part	Description
`p`	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch). <*n*> is the level at which the module is set to trigger in the **Define Steps** or **Modify Steps** window.
`StepName`	The name of the module step as defined in the **Define Steps** window.

Comments

Use the `Prepare` event handler to initialize a task. It occurs immediately before a task is sent to a module. The "Inserting an Event Handler" on page 515 section contains information about creating the "shell" of a `Prepare` event handler.

Retrigger Event

A module can use the `Retrigger` event handler to retrigger a task. The retriggering request is made using the client *API* which tells the Intelligent Capture Server to run the `Retrigger` function in the *IPP*. If the function does not exist in the *IPP*, then a standard method for retriggering is used. Furthermore, if the function returns `IA_ERR_NOFUNC (-4518)`, then the standard method is used.

Note: To use the `Retrigger` event handler, you must have a module that has been designed to call the `IABatchRetriggerNode` client *API* function. For example, when you retrigger a step from Intelligent Capture Administrator, it calls the `Retrigger` event handler for every task-level node in the tree. The `Retrigger` event handler is a function, not a subroutine. In other words, it can return a value to the Intelligent Capture Server.

Syntax

```
Private Function <Stepname>_Retrigger(ByVal <p> As IAS_RECORD_<n>, ByVal <pInfo> As
IIAS_Info) As Long
```

The `Retrigger` event handler syntax has the following parts:

Table 8-36: `Retrigger` Event Handler Syntax

Part	Description
`p`	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of Level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch). <*n*> is the level at which the module is set to trigger in the **Define Steps** or **Modify Steps** window.
`StepName`	The name of the module step as defined in the **Define Steps** window.
`pInfo`	An object of type `IIAS_Info` containing information about who called the `Notify` event handler. The "`IIAS_Info Object`" on page 555 section contains more information.

Based on the return value of this function, the standard method of retriggering is used in addition to the code already run in the event handler.

Comments

The `Retrigger` event handler enables the *IPP* to decide dynamically to retrigger a task. For example, if you retrigger a step from Intelligent Capture Administrator, then all task-level nodes are retriggered. With the `Retrigger` event handler, your *IPP* can intercept the request to retrigger each node, and dynamically decide which nodes to retrigger and which to ignore.

If your implementation of this function returns `IA_ERR_NOFUNC (-4518)`, then the standard method of retriggering also can be used. Therefore, your process can record information when tasks are retriggered, but let the Intelligent Capture Server take care of the actual retriggering of the tasks.

The following characteristics of the `Retrigger` event handler set it apart from the `Error`, `Finish`, and `Prepare` event handlers. A module can call the `Retrigger` event handler:

- Without having received a task.
- For any task-level node in the tree. Nodes can be task nodes processed by another module, nodes that are not currently being processed, or even nodes not ready for processing.

  Note: If you retrigger tasks from Intelligent Capture Administrator, then the Intelligent Capture Server calls the `Retrigger` event for every task-level node in the tree.

- For any step in the *IPP*. The step can be a step of the calling module or not.

IA Values

This section provides reference information for IA Values that can be used in an *IPP*.

Reserved IA Values

The following table lists IA values that are reserved. Reserved IA values have special meaning inside the Intelligent Capture Server process code (even though they are not defined in an *MDF*). Do not redefine the variables using these names within an *IPP* or *MDF*. If you want to access or manipulate any of the following values, assign them to another variable.

Table 8-37: Reserved IA Values

IA Value	Description
<Any>	Used to access common member variables in an `IAVariant` data type. **Note:** `IAVariant` is supported in Process Developer, but it is not supported in Intelligent Capture Designer.
<BatchHold>	Places a batch in a hold state.
<BatchError>	Places a batch in an error state.
<BatchID>	Identification number assigned to a batch by the Intelligent Capture Server.
<Batchname>	Name assigned to a batch when it is created.
<Compile_AppVersion>	Version of Process Developer used to compile a process.
<Compile_Time>	Time that Process Developer compiled a process. The format of this value is dependent on the date/time/internalization settings of the Intelligent Capture Server.
<Compile_VbaVersion>	Version of VBA that Process Developer used to create an *IPP* file.

IA Value	Description
<Description>	Description of the batch, as entered or modified by a ScanPlus or Intelligent Capture Administrator operator.
<ErrorStatus>	Places a task in an error state.
<IAComDLLVersion>	Internal version number of the `IAComDLL` currently installed on the Intelligent Capture Server.
<IADepartment>	Department name that is associated with a step in an *IPP* file.
<IATaskRouting>	Department name that is associated with a task in an IPP file.
<Item>	Used to access dynamic values.
<LockDebug>	Internal value that indicates the lock status of the node being processed.
<Priority>	Priority of a batch (default 50).
<ProcessCompilerVersion>	Internal number of the compiler used to create the process. Version 3.0 of Process Developer returns 10.
<ProcessName>	Name of the process from which a batch was derived.
<Ready>	Trigger IA value used to indicate that the specified node is ready for processing.
<RetriesLeft>	Special counter IA value controlling how many times a task that encounters an error should be reprocessed by the module where the error occurred.
<Status>	Processing status of a specified node. Possible values include: • 0: Not applicable • 1: Not ready • 2: Ready • 3: Working • 4: Done
<TaskCount>	Number of pending tasks in this batch for the specified step.
<TaskCountA>	Number of active tasks in this batch for the specified step.
<TaskCountQ>	Number of queued tasks in this batch for the specified step.

IA Value	Description
<Type>	The name of the object type of an Object, or the name of the currently active object type of an IAVariant. **Note:** `IAVariant` is supported in Process Developer, but it is not supported in Intelligent Capture Designer.
<TreeVersion>	Local Intelligent Capture Server version number for theIntelligent Capture tree, used internally to keep the tree synchronized.
<Trigger_*>	Reserved value where "*" is wild card.
<TriggerLevel>	The trigger level of the step as specified in the IPP and XPP files.

BatchHold IA Value

Sets a batch in a hold state.

Syntax

```
<$batch>|<BatchHold>
```

Table 8-38: <BatchHold> Syntax

Part	Description
`$batch`	A non-nodal, key string value that identifies the batch to set.

Comments

Setting the *<BatchHold>* status can be used for placing a batch on hold, instead of setting the batch priority to 0. By default, the *<BatchHold>* status is disabled. When *<BatchHold>*=1, tasks are not sent to modules for processing. This IA Value can be set at any time. It does not stop tasks that have already been sent for processing.

Example

This example uses `IAValueSetLong` to set the *<BatchHold>* status:

```
Call IAValueSetLong("$batch=" & p.Tree.BatchID & "/BatchHold",1)
```

This example uses `IAValueSetAscii` to set the *<BatchHold>* status:

```
Call IAValueSetAscii("$batch=" & p.Tree.BatchID & "/BatchHold","1")
```

BatchError IA Value

Sets a batch in an error state.

Syntax

```
<$batch>|<BatchError>
```

Table 8-39: <BatchError> Syntax

Part	Description
`$batch`	A non-nodal, key string value that identifies the batch to set.

Comments

Setting the <*BatchError*> status can be used for handling errors, instead of setting the batch priority to 0. By default, the <*BatchError*> status is disabled. When <*BatchError*>=1, batches and tasks are not sent to modules for processing. When the error state is cleared, task processing is resumed.

Example

This example uses `IAValueSetLong` to set the <*BatchError*> status:

```
Call IAValueSetLong("$batch=" & p.Tree.BatchID & "/BatchError",1)
```

This example uses `IAValueSetAscii` to set the <*BatchError*> status:

```
Call IAValueSetAscii("$batch=" & p.Tree.BatchID & "/BatchError","1")
```

ErrorStatus IA Value

Places a task in a task error state.

Syntax

```
<$batch>|<$step>|<$node>|<ErrorStatus>
```

Table 8-40: <ErrorStatus> Syntax

Part	Description
`$batch`	A non-nodal, key string value that identifies the batch to set.
`$step`	A value that identifies the step to set. <*$step*> can not be used without using <*$batch*>.
`$node`	MDF value or dynamic value in the batch tree that identifies the node to be set. <*$node*> can not be used without using <*$batch*> and <*$step*>.

Comments

Setting the <*ErrorStatus*> can be used for handling errors, instead of setting <*RetriesLeft*> to 0. By default, the <*ErrorStatus*> is disabled. When <*ErrorStatus*>=1,

tasks are not sent to modules for processing. When the <*ErrorStatus*> is cleared, task processing resumes.

Example

This example uses `IAValueSetLong` to set the <*ErrorStatus*> status:

```
Call IAValueSetLong("$batch=" & p.Tree.BatchID & "/$step=StepName/$node=" &
p.Tree.NodeID & "/ErrorStatus",1)
```

This example uses `IAValueSetAscii` to set the <*ErrorStatus*> status:

```
Call IAValueSetAscii("$batch=" & p.Tree.BatchID & "/$step=StepName/$node=" &
p.Tree.NodeID & "/ErrorStatus","1")
```

Compile_TriggerState IA Value

A batch IA Value that reflects whether this batch has been set up with automatic retriggering enabled.

Syntax

```
<lRetrigger> = IAValueGetLong("$batch=<batchid>/Compile_TriggerState", 0)
```

The Compile_TriggerState property syntax has the following parts:

Table 8-41: <Compile_TriggerState> Syntax

Part	Description
<*batchid*>	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch).
`lRetrigger`	A return value of type Long, indicating for what type of retriggering this batch is configured: • 0 (default): Automatic retriggering is enabled. • 1: Manual retriggering is enabled.

Comments

Use this IA Value when you save an *IPP* from an existing batch using the **Save As** window. This way you can ensure that the new *IPP* correctly preserves the state of the automatic retriggering option.

Ready Trigger IA Value

<*Ready*> is a trigger IA Value defined by Process Developer for all steps at every level.

Syntax

```
<p>.<StepName>.Ready = <iGo>
```

The <*Ready*> IA Value syntax has the following parts:

Table 8-42: <Ready> Trigger IA Value

Part	Description
`p`	An object of type `IASLib.IAS_RECORD_`*<n>* containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch).
`StepName`	The name of the module step as defined in the **Define Steps** window.
`iGo`	A variable of type Integer. Zero indicates not ready; non-zero indicates ready.

Comments

<*Ready*> is a trigger IA Value of type Long that Process Developer defines for each step at every level. <*Ready*> works in the same way as other triggers:

- <*Ready*> only acts as a trigger if referenced. Otherwise it is ignored.
- If it is referenced and set to a non-zero value, the module step (specified in <*StepName*>) starts processing when all other trigger conditions have been satisfied.
- If it is referenced and is set to zero, the module cannot begin processing tasks, regardless of the status of other trigger conditions.

Note: The <*Ready*> IA Value cannot be declared in an *MDF*.

Example

In this example, when <*Ready*> is set to 1, it triggers the Copy module.

```
Private Sub Scan_Finish (ByVal p As IAS_RECORD_0)
    p.ImageExp.InputImage = p.Scan.OutputImage
    ' Trigger the Copy utility by setting Ready to a non-zero value:
    p.Copy.Ready = 1
End Sub
```

RetriesLeft IA Value

<*RetriesLeft*> is a special IA Value defined by Process Developer for all steps at the trigger level. It can be used to control the behavior of a task when a module returns an error. (The Ready trigger contains additional information.)

Syntax

```
<p>.<StepName>.RetriesLeft
```

The <*RetriesLeft*> IA Value syntax has the following parts:

Table 8-43: <RetriesLeft> IA Value

Part	Description
`p`	An object of type `IASLib.IAS_RECORD_`*<n>* containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch).
`StepName`	The name of the module step as defined in the **Define Steps** window.

Comments

<*RetriesLeft*> is an IA Value that Process Developer automatically declares at the trigger level of each step. It is available even if it is not referenced in the *IPP*. The default value is 3, though it can be changed to another value at any point in the process (for example, in the `Process_Install` event handler). It specifies how many times the Intelligent Capture Server resubmits a task with an error result to a module before giving up.

If the step does not have an `Error` event handler, a default error handler is called. The default behavior of this error routine is that the server does not automatically decrement <*Retriesleft*> any lower than 1. The task gets its error flag set and <*RetriesLeft*> remains at 1. Therefore, the task is still triggered and clearing the error state restarts the task.

If the step has an `Error` event handler, the Server calls that routine instead of the default error routine. It is up to the user to decide the behavior of the error routine. The routine can be to have the Server decrement <*RetriesLeft*> down to 0, or to set the batch priority to 0.

The recommended behavior for an error routine is to emulate what the Intelligent Capture Server does. From the *IPP* code, it is possible to decrement <*RetriesLeft*> down to 1. If <*RetriesLeft*> is already 1 and the error routine has been called again, set the task error flag. Set the task error flag using `IAValueSetLong`. Or, the *IPP* code can set the <*BatchError*> flag, and then reset <*RetriesLeft*> to 3.

Note: The <*RetriesLeft*> IA Value cannot be declared in an *MDF*.

Methods

IAValueGet/Set Methods

The `IAValueGetAscii`, `IAValueSetAscii`, `IAValueGetLong`, and `IAValueSetLong` functions enable you to access data stored as IA Values without having to traverse the batch tree first.

Using the strKey Argument

To use IAValueGet/Set methods, construct a key string (referred to as `strKey`) to identify the IA Value you want to get or set. This key string consists of one or more special strings and an IA Value name (the value you want to get or set).

Special Strings for Constructing strKey

You can use some or all of the following special strings to focus the scope of where your IA Value is located:

- *`$batch` values (also called per-batch IA Values)*: Values created for setting up your batch. These values are also referred to as non-nodal values. `$batch` must be set to a valid batch (or process) ID in decimal notation (as opposed to hexadecimal). For example:

  ```
  "$batch=2477/LevelFormat_0"
  ```

- *`$step` values (also called per-step IA Values)*: Values created when you set up each process step. You cannot use `$step` without using `$batch`. You can also access a step value without specifying `$step`. You can use the `StepName.Value` syntax instead. For example:

  ```
  "$batch=2477/$step=Scan/ExpectedPageCount" "$batch=2477/Scan.ExpectedPageCount"
  ```

- *`$node` values*: *MDF* values and dynamic values on the individual nodes in the batch tree. You cannot use `$node` without using both `$step` and `$batch` (because nodal values must be associated with a step in a batch). For example:

  ```
  "$batch=2477/$node=379/$step=Scan/ImageBytes"
  "$batch=2477/$node=379/Scan.ImageBytes"
  ```

To join special strings, you can concatenate them like you would in VB string construction:

```
strKey = "$batch=" & p.Tree.BatchID & "/$step=StepName/$node=" & p.Tree.NodeID & "/
<ValueName>"
```

The special strings are case sensitive (`"$Batch"` is not valid). The order in which you construct them does not matter, however.

Constructing the IA Value Key

The IA Value key is the name of the value you are trying to get or set. It does not include the special strings that identify the batch, node, or step associated with that value. For example, in the following `strKey` strings, the emphasized portion is the key:

```
"$batch=123/$node=456/$step=Scan/ImageBytes" "$batch=234/$node=345/Scan.ImageBytes"
"$batch=100/$node=200/Custom.MyFriend:Athlete:Sport"
```

Note: If you use the `StepName.Value` syntax, then only the portion of the string after `"StepName."` is considered the key. If you are using Objects or IAVariants, the parent object, child IA Values, and the ':' (colon) separators are also included in the key. The `strKey` argument must not exceed a length of 255 characters.

Using IAValueGet/Set Methods with Objects and IAVariants

`IAValueGet/Set` methods cannot directly access an `Object` or `IAVariant`, but must access member variables. To access member variables of an Object, use the following syntax:

```
strTemp = IAValueGetAscii( "$batch=" & ... & "/<ObjectVariable>:<MemberVariable>",
"default")
```

Where:

- *<ObjectVariable>* specifies an IA Value of type Object.
- *<MemberVariable>* is one of the member variables of the Object.
- ":" (colon) separates the Object variable name from the reference to the member variable name.

The following is an example of a value key string that provides addressing into an Object IA Value called `Worker` with a member variable `Age`:

```
"$batch=234/$node=234/$step=Custom/Worker:Age"
```

The "Object" on page 492 and "IAVariant" on page 496 sections contain more information on using `IAValueGet/Set` methods with Objects and IAVariants.

GetDataProperty Method

Returns the value of the data property of a given name in a given string.

Syntax

```
<strValue> = GetDataProperty(<Str>, <Property>)
```

The `GetDataProperty` method syntax has the following parts:

Table 8-44: GetDataProperty Method

Part	Description
`Str`	A variable of type String, the list of data properties, delimited by newline characters (represented here by the string "\n".)
`Property`	A variable of type String, the name of the data property to search for.
`strValue`	A variable of type String, the value of the requested data property.

Comments

Searches `Str`, looking for a substring that looks like the following: "*<Property>*`=somevalue\n"` (if the end of the string is reached, then the newline character is optional.) If a matching substring is found, then the portion of the

substring between the '=' and the '\n' is returned. If no matching substring is found, an empty string is returned.

Example

This example gets the `Version` property of a string, which returns “6.5”. It also looks for the `ServicePack` property, which returns an empty string.

```
Option Explicit

Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim Props As String
    Dim Version As String
    Dim ServicePack As String

    Props = "Product=InputAccel" & vbLf & "Version=6.5"
    Version = GetDataProperty(Props, "Version")
    ServicePack = GetDataProperty(Props, "ServicePack")
End Sub
```

`GetField` Method

Returns the <*n*>th field (substring) in a delimited string.

Syntax

```
strValue = GetField(Str, FieldNum, Separator)
```

Table 8-45: Parameters

Value	Description
Str	A variable of type String, the list of fields, divided by separator characters.
FieldNum	A variable of type Long, the number of the field to return, starting with 1.
Separator	A variable of type String, a set of characters that can act as delimiters to separate each field.
StrValue	A return value of type String, the value of the requested field.

Comments

If multiple separator characters are specified, then any of those characters can separate a field. If *FieldNum* is greater than the number of fields in the string, then separators are added to ensure the correct number of fields. The new separators added, if any, are the first character in *Separator*.

StrField does not need to be the same length as the field it is replacing.

Example

This example sets the fourth field in a string, changing "Mary" to "Simon".

```
Option Explicit

Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim temp As String
    Dim field As String
    temp = "John,Jason;Michael-Mary,Ruth"
    temp = GetField(temp, 4, ",;-")
End Sub
```

This example gets the “Version” property of a string, which returns “6.5”. It also looks for the “ServicePack” property, which returns an empty string.

```
Option Explicit

Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim Props As String
    Dim Version As String
    Dim ServicePack As String
    Props = "Product=InputAccel" & vbLf & "Version=6.5
    Version = GetDataProperty(Props, "Version")
    ServicePack = GetDataProperty(Props, "ServicePack")
End Sub
```

`IALogDebug` Method

The `IALogDebug` method writes messages to the `debug.out` log file in the *IAS* directory of the Intelligent Capture Server.

Syntax

```
Call IALogDebug(<dwBits>, <strLog>)
```

The `IALogDebug` event syntax has the following parts:

Table 8-46: IALogDebug Event Syntax

Part	Description
`dwBits`	A variable of type Long, the bit mask that determines whether or not the Intelligent Capture Server log this entry.
`strMessage`	A variable of type String, the message to write to the debug.out log file.

Comments

The `IALogDebug` method can log an entry to the `debug.out` log file of the Intelligent Capture Server. `dwBits` determines whether the entry is logged. The value of `dwBits` corresponds to the server setting, `FileTraceLevel`. The server setting `FileTraceLevel` is used to determine which events are logged.

To determine if an entry is logged, do a bitwise AND of the `dwBits` value and the `FileTraceLevel` value. If the result is non-zero (that is, the two values have at least 1 bit in common that is set), then the entry is logged.

By default, FileTraceLevel is set to &H70 (112 in decimal), which is &H40+&H20 + &H10. In this case, only warnings (&H10), errors (&H20), and fatal errors (&H40) are logged to the debug.out log file. For more information on the FileTraceLevel server parameter, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

Two bits have been reserved for use by *IPP* writers to use in the IALogDebug method. Bits &H200 and &H400 are reserved exclusively for *IPP* developers. If you want to view these events, modify your FileTraceLevel server setting. The easiest way, is from Intelligent Capture Administrator. A value of &H670 (decimal 1648) would enable these two reserved bits, while otherwise maintaining the default value of &H70. The special bit &H80000000 has been reserved for flushing the debug.out file to disk. When theIntelligent Capture Server is busy, it cannot write the contents of the debug.out file to disk immediately. By using this bit, you can force the Intelligent Capture Server to flush the log to disk immediately. This way, if the Intelligent Capture Server crashes on the next operation, then the log entry is preserved. For example, a value of dwBits = &H80000220 indicates an error (&H20), logged by the *IPP* (&H200), and it is logged to disk immediately (&H80000000).

Example

This example logs tree manipulation events to the debug.out file. Log such events to debug an error during tree manipulation when you are not sure when the error occurs or what module or user is causing it.

The PostNodeAdd1 and PostNodeMove0 events are shown. These events are called when a document is split. The PreNodeMove0 event handler is also called and it is almost identical to PostNodeMove0.

```
Option Explicit

Private Sub Tree_PostNodeAdd1(ByVal p1 As IASLib.IAS_RERD_1, ByVal pInfo As IIAS_Info)
    Call IALogDebug(&H200, "PostNodeAdd1( " & p1.Tree.NodeID & " )
        called from " & pInfo.Module & " by user '" & pInfo.User & "'")
End Sub

Private Sub Tree_PostNodeMove0(ByVal pDest As IASLib.IAS_RECORD_1, ByVal pSrc As
IASLib.IAS_RECORD_1,
        ByVal pNodes As IASLib.PageCollection, ByVal pInfo As IIAS_Info)
    Dim p0 As IASLib.IAS_RECORD_0
    Dim nodes As String

    nodes = " "
    For Each p0 In pNodes
        If nodes <> "" Then nodes = nodes & " " nodes = nodes & p0.Tree.NodeID
    Next
    nodes = "<" & nodes & ">"
    Call IALogDebug(&H200, "PostNodeMove0( " & pDest.Tree.NodeID & ", " &
pSrc.Tree.NodeID & ",
         " & nodes & ") called from " & pInfo.Module & " by user '" & pInfo.User & "'")
End Sub
```

IALogNTMessage Method

The IALogNTMessage method writes messages to the Windows Application Event Log on the computer hosting the Intelligent Capture Server.

Syntax

```
Call IALogNTMessage(<iType>, <strMessage>)
```

The `IALogNTMessage` event syntax has the following parts:

Table 8-47: IALogNTMessage Event Syntax

Part	Description
`iType`	A variable of type Integer, one of the error type constants that you must define in a non-class Visual Basic module.
`strMessage`	A variable of type String, the message to write to the Event Log.

Comments

The `IALogNTMessage` method can log an event to the Intelligent Capture Server. *<iType>* determines the type of event logged. It is one of the following error constants declared in the Common_Constants module:

```
Public Const IALOG_SUCCESS = &H80
Public Const IALOG_INFO = &H4
Public Const IALOG_WARNING = &H2
Public Const IALOG_ERROR = &H1
Public Const IALOG_AUDITSUCCESS = &H8
Public Const IALOG_AUDITFAILURE = &H10
```

Each of these constants, except `IALOG_SUCCESS`, corresponds to the event of the same type. (For example, `IALOG_ERROR` corresponds to an `Error` event.) The server setting of the Intelligent Capture Server `EventLogLevel` is used to determine the events that are logged. Set bits correspond to events that be logged. By default, `EventLogLevel` is set to `IALOG_SUCCESS + IALOG_WARNING +IALOG_ERROR = &H83`.

`IALOG_SUCCESS` and `IALOG_INFO` both log as Information, but they can be independently turned on and off. `IALOG_INFO` is used mainly for debugging information on the Intelligent Capture Server and end users normally do not use it. `IALOG_SUCCESS` is used for information such as “server started”, “server stopped”, and “server paused”. Writers of *IPP*s are free to select the event that suits their needs.

Note: Include code to log errors to the Windows Event Log. Logging errors enable you to determine, after the fact, whether errors occurred in a batch and what the errors were.

Example

This example reports to the system Event Log a batch that has had an error in processing:

```
Option Explicit

Private Sub Scan_Error(ByVal p As IASLib.IAS_RECORD_O, ByVal Code As Integer)
    Call IALogNTMessage(IALOG_WARNING, "Scan_Error called in batch " &
       p.Tree.BatchName & ", Error code " & Code)

End Sub
```

IAValueGetAscii Method

Returns the specified IA Value as a string.

Syntax

```
<strValue> = IAValueGetAscii(<strKey>, <strDefault>)
```

The `IAValueGetAscii` method syntax has the following parts:

Table 8-48: `IAValueGetAscii` Method Syntax

Part	Description
`strKey`	A variable of type *String*, the key string of the value you want to get.
`strDefault`	A variable of type *String*, a default value you specify that is used if no value can be found.
`strValue`	A return value of type *String*, the requested value.

Comments

Call the `IAValueGetAscii` method to retrieve batch values. You can get IA Values from any node and step in the batch. You can also get per-step IA Values and per-batch IA Values. If no value has previously been set for the key, then the method gets the default value you specify in `strDefault`.

To use `IAValueGetAscii`, create a key string, `strKey`. The "IAValueGet/Set Methods" on page 576 contains more information.

Example

This example gets the per-batch IA Value, `LevelName_O`. If this variable does not exist, then the `IAValueGetAscii` method returns `"Page"`.

```
Option Explicit

Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_O)
    Dim strLevelOName As String
    strLevelOName = IAValueGetAscii("$batch=" & p.Tree.BatchID & "/LevelName_O", "Page")
End Sub
```

IAValueGetLong Method

Returns the specified IA Value as a long value.

Syntax

```
<lValue> = IAValueGetLong(<strKey>, <lDefault>)
```

The `IAValueGetLong` method syntax has the following parts:

Table 8-49: `IAValueGetLong` Method Syntax

Part	Description
`strKey`	A variable of type String, the key string of the value you want to get.
`lDefault`	A variable of type Long, a default value you specify that is used if no value can be found.
`lValue`	A return value of type Long, the requested value.

Comments

Call the `IAValueGetLong` method to retrieve batch values. You can get IA Values from any node and step in the batch; you can also get per-step IA Values and per-batch IA Values. If no value has previously been set for the key, then the method gets the default value you specify in `lDefault`.

To use `IAValueGetLong`, create a key string, `strKey`.

Example

This example gets the per-batch IA Value `LevelVisible_3`. If this variable does not exist, then the `IAValueGetLong` method returns the value 0.

```
Option Explicit

Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_O)
    Dim LevelVisible As Long
    LevelVisible = IAValueGetLong("$batch=" & p.Tree.BatchID & "/LevelVisible_3", 0)
End Sub
```

Related Topics

"IAValueGet/Set Methods" on page 576

IAValueSetAscii Method

Sets the specified IA Value.

Syntax

```
Call IAValueSetAscii(<strKey>, <strValue>)
```

The `IAValueSetAscii` method syntax has the following parts:

Table 8-50: `IAValueSetAscii` Method Syntax

Part	Description
`strKey`	A variable of type String, the key string of the value you want to get.
`strValue`	A return value of type String, the value to set.

Comments

Call the `IAValueSetAscii` method to set batch values. You can set IA Values for any node and step in the batch; you can also set per-step IA Values and per-batch IA Values. We do not recommend that you use `IAValueSetAscii` to set a value longer than the supported Visual Basic limit for fixed-length strings (65,400).

To use `IAValueSetAscii`, create a key string, <*strKey*>. To use `IAValueGetAscii`, create a key string, <*strKey*>. The "IAValueGet/Set Methods" on page 576 contains more information.

Example

This example sets the per-batch IA Value `LevelFormat_O` to the string `"page @O2"`, which sets the display name printed under thumbnails in the tree.

```
Option Explicit

Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_O)
    Call IAValueSetAscii("$batch=" & p.Tree.BatchID & "/LevelFormat_O", "page @O2")
End Sub
```

IAValueSetLong Method

Sets the specified IA Value.

Syntax

```
Call IAValueSetLong(<strKey>, <strValue>)
```

The `IAValueSetLong` method syntax has the following parts:

Table 8-51: `IAValueSetLong` Method Syntax

Part	Description
`strKey`	A variable of type String, the key string of the value you want to get.
`lValue`	A return value of type Long, the value to set.

Comments

Call the `IAValueSetLong` method to set batch values. You can set IA Values for any node and step in the batch; you can also set per-step IA Values and per-batch IA Values.

When using `IAValueSetLong`, create the key string `strKey`. The "IAValueGet/Set Methods" on page 576 section contains more information.

Note: Within *IPP*, the `IAValueSetLong` method no longer returns error values. You must trap errors in other ways, such as by checking the `Number` property of the `Err` object.

Example

This example sets the per-batch IA Value `LevelVisible_3` to the value 1. Therefore, level 3 nodes is visible in the tree.

```
Option Explicit
Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_O)
    Call IAValueSetLong("$batch=" & p.Tree.BatchID & "/LevelVisible_3", 1)
End Sub
```

SetDataProperty Method

Sets the value of the data property of a given name in a given string, returning the modified string.

Syntax

```
<strValue> = SetDataProperty(<Str>, <Property>, <Value>)
```

The `SetDataProperty` method syntax has the following parts:

Table 8-52: `SetDataProperty` Method Syntax

Part	Description
`Str`	A variable of type String, the list of data properties, delimited by newline characters (represented here by the string `"\n"`.)
`Property`	A variable of type String, the name of the data property to search for.
`Value`	A variable of type String, the new value for the data property with the name *<Property>*.

Part	Description
`strValue`	A variable of type String. The return value of this function is the string `Str` with the modified or added data property *<Property>*=*<Value>*. The variable `Str` is not changed (unless you set it equal to the return value of this function).

Comments

Searches `Str`, looking for a substring that looks like the following: "*<Property>*=somevalue\n". If the end of the string is reached, then the newline character is optional. If a matching substring is found, then *<Value>* replaces the portion of the substring between the '=' and the '\n'. For example, "*<Property>*=somevalue\n" is changed to "*<Property>*=*<Value>*\n". If no match is found, then the string "*<Property>*=*<Value>*\n" is appended to `Str` and returned).

Example

This example sets the "Version" and "ServicePack" properties of a string. Therefore, the "ServicePack" data property is appended to the string.

```
Option Explicit

Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim Props As String
    Props = "Product=InputAccel" & vbLf & "Version=5.0"
    Props = SetDataProperty(Props, "Version", "4.0")
    Props = SetDataProperty(Props, "ServicePack", "4")
    ' At this point, Props =
    ' "Product=InputAccel\nVersion=4.0\nServicePack=4\n"
End Sub
```

SetField Method

Sets the <*n*>th field (substring) in a delimited String, returning the modified string.

Syntax

```
<strValue> = SetField(<Str>, <FieldNum>, FieldStr, Separator)
```

The `SetField` method syntax has the following parts:

Table 8-53: `SetField` Method Syntax

Part	Description
`Str`	A variable of type String, the list of fields, divided by separator characters.
`FieldNum`	A variable of type Long, the number of the field to return, starting with 1.

Part	Description
`FieldStr`	A variable of type String, the new field to insert into Str.
`Separator`	A variable of type String, a set of characters that can act as delimiters to separate each field.
`strValue`	A variable of type String. The return value of this function is the string `Str` with the field `FieldStr` inserted. The variable `Str` is not changed (unless you set it equal to the return value of this function).

Comments

If multiple separator characters are specified, then any of those characters can separate a field. If `FieldNum` is greater than the number of fields in the string, then separators are added to ensure the correct number of fields. The new separators added, if any, is the first character in `Separator`.

`StrField` does not need to be the same length as the field it is replacing.

Example

This example sets the fourth field in a string, changing “Mary” to “Simon”.

```
Option Explicit
Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim temp As String
    Dim field As String

    temp = "John,Jason,Michael,Mary,Ruth"
    temp = SetField(temp, 4, "Simon", ",")
End Sub
```

Properties

This section describes the properties you can use to perform tree navigation within your processes.

Batch Property

Returns the batch ID of the current batch. This property is available for backward compatibility only. Use instead the new BatchID property.

Syntax

```
<lBatchID> = <p>.Tree.Batch
```

The Batch property syntax has the following parts:

Table 8-54: Batch Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_n containing the record structure of level n, where n is an integer from 0 (page) through 7 (batch).
lBatchID	A return value of type *Long*, the ID of the current batch.

Comments

The Batch property is identical to the BatchID property. It is recommended that you use the BatchID property because its name is more descriptive, and because its future compatibility is guaranteed. The Batch property is read-only.

BatchID Property

Returns the batch ID of the current batch.

Syntax

```
<lBatchID> = <p>.Tree.BatchID
```

The BatchID property syntax has the following parts:

Table 8-55: BatchID Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_*<n>* containing the record structure of level n, where n is an integer from 0 (page) through 7 (batch).
lBatchID	A return value of type Long, the ID of the current batch.

Comments

The BatchID property retrieves the batch ID from a root level node of the batch. It is useful when constructing keys for IAValueGet/Set methods), in which you often use a batch ID within a key string. The BatchID property is read-only.

Example

This example uses BatchID with the IAValueGetAscii method to create a string that includes the batch ID of the current batch. In addition, IAValueGetAscii accesses the per-batch IA Value LevelName_0.

```
Option Explicit

Private Sub Scan_Finish(ByVal p As IASLib.IAS_RECORD_0)
```

```
Dim strLevel0Name As String

    strLevel0Name = IAValueGetAscii("$batch=" & p.Tree.BatchID & "/LevelName_0", "Page")
End Sub
```

BatchName Property

Returns the name of the current batch.

Syntax

```
<strBatchName> = <p>.Tree.BatchName
```

The BatchName property syntax has the following parts:

Table 8-56: BatchName Property Syntax

Part	Description
p	An object of type `IASLib.IAS_RECORD_n` containing the record structure of Level n, where n is an integer from 0 (page) through 7 (batch).
`strBatchName`	A return value of type String, the name of the current batch.

Comments

The BatchName property retrieves the name of the current batch. (You are no longer required to use the `IAValueGetAscii` method to get the value of the key string `"$batch/BatchName"`.) While the old way of getting the batch name is still supported, using the BatchName property is simpler and results in easier-to-understand *IPP*.

Example

This example reports to the system Event Log which batch has finished scanning.

```
Option Explicit

Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Call IALogNTMessage(IALOG_INFO, "ScanPlus_Finish called in Batch "
        & pRoot.Tree.BatchName)
End Sub
```

Child Property

Returns the specified child node object.

Syntax

```
Set <pNode> = <p>.Tree.Child(<lIndex>)
```

The Child property syntax has the following parts:

Table 8-57: Child Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_n containing the record structure of level n, where n is n integer from 1 (document) through 7 (batch).
lIndex	A variable of type Long, the index number of the desired child node. (0-based.)
pNode	A return value of type IAS_RECORD_<*m*>, the node object of the child node, where <*m*> is one less than the level at which the Child property executes.

Comments

The Child property returns the node belonging to the index number you specify. *lIndex* is 0-based, so legal index values range from 0 to (number of children) - 1. The index count begins on the level represented by <*n*>. If successful, the property returns the node object in *pNode*. The Child property is read-only.

This property is not available for level 0 nodes, because they do not have children.

Note: Within the *IPP*, the Child returns a node reference object. It does not return an error code. You must trap errors in other ways, such as by checking the Number property of the Err object.

Example

This example loops through all the children of the task node, creating a dynamic value named NewVal (which stores the value 12) for each child node:

```
Option Explicit

Private Sub IndexExp_Finish(ByVal p1 As IASLib.IAS_RECORD_1)
    Dim Index As Long
    Dim Count As Long
    Dim pChild As IASLib.IAS_RECORD_0

    Count = p1.Tree.NumChildren(0)
    For Index = 0 To Count - 1
        Set pChild = p1.Tree.Child(Index)
        pChild.IndexExp("NewVal") = "12"
    Next
End Sub
```

Children Property

Returns a collection of children nodes, which can be accessed in a For Each loop

Syntax

```
For Each <pChild> in <p>.Tree.Children
```

The Children property syntax has the following parts:

Table 8-58: Children Property Syntax

Part	Description
<p>	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level *<n>*, where *<n>* is an integer from 1 (document) through 7 (batch).
<pChild>	A return value of type `IAS_RECORD_<m>`, the node object of the child node, where *<m>* is one less than the level at which the `Child` property executes.

Comments

The Children property returns a collection object, which is used to loop through all of the child nodes of the object. Compared to using the Child property, the Children property usually simplifies the code required when performing operations on each child of a node. The Children property is read-only and can exist only for level 1 and higher nodes.

Example

This example loops through each child node using the Children property, creating a dynamic value named `NewVal` (which stores the value 12) for each child node. Compare this example to the example using the Child property to view how the Children property can be easier to use.

```
Option Explicit

Private Sub IndexExp_Finish(ByVal p1 As IASLib.IAS_RECORD_1)
    Dim pChild As IASLib.IAS_RECORD_0
    For Each pChild In p1.Tree.Children
        pChild.IndexExp("NewVal") = 12
    Next
End Sub
```

Related Topics

"Child Property" on page 590

L1Child Property

The L1Child property returns level 1 descendant node objects of a level 3, 4, 5, 6 or 7 node.

Syntax

```
Set <pNode> = <p>.Tree.L1Child(<lIndex>)
```

The L1Child property syntax has the following parts:

Table 8-59: L1Child Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_*<n>* containing the record structure of level <*n*>, where n is n integer from 3 through 7.
lIndex	A variable of type Long, the index number of the desired level 1 node. (0-based.)
pNode	A return value of type IAS_RECORD_1, the node object of the level 1 child node.

Comments

Use the L1Child property to get all of the level 1 descendant node objects of the current node without first traversing the tree. lIndex is 0-based, so legal index values range from 0 to (number of children) - 1. The L1Child property is read-only.

This property is only available for level 3 through 7 nodes. For level 2 nodes, use the Child property. Level 1 nodes do not have level 1 children.

Example

This example loops through all the level 1 children of the current task node:

```
Option Explicit

Private Sub IndexExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim Index As Long
    Dim Count As Long
    Dim pL1Child As IASLib.IAS_RECORD_1

    Count = pRoot.Tree.NumChildren(1)

    For Index = 0 To Count - 1

        Set pL1Child = pRoot.Tree.L1Child(Index)
        'Add your code here which performs some function on
        'each pL1Child
        Next
End Sub
```

L2Child Property

Returns level 2 descendant node objects of a level 4, 5, 6 or 7 node.

Syntax

```
Set <pNode> = <p>.Tree.L2Child(<lIndex>)
```

The L2Child property syntax has the following parts:

Table 8-60: L2Child Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_*<n>* containing the record structure of Level <*n*>, where n is n integer from 4 through 7.
lIndex	A variable of type Long, the index number of the desired level 2 node. (0-based.)
pNode	A return value of type IAS_RECORD_2, the node object of the level 2 child node specified by lIndex.

Comments

Use the L2Child property to get all of the level 2 descendant node objects of the current node without first traversing the tree. *lIndex* is 0-based, so legal index values range from 0 to (number of children) - 1. The L2Child property is read-only.

This property is only available for level 4 through 7 nodes. For Level 3 nodes, use the Child property. Level 1 and level 2 nodes do not have level 2 children.

Example

This example loops through all the level 2 children of the current task node:

```
Option Explicit

Private Sub IndexExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim Index As Long
    Dim Count As Long
    Dim pL2Child As IASLib.IAS_RECORD_1

    Count = pRoot.Tree.NumChildren(2)
    For Index = 0 To Count - 1
        Set pL2Child = pRoot.Tree.L2Child(Index)
        'Add your code here which performs some function on
        'each pL1Child
    Next
End Sub
```

L2Parent Property

The L2Parent property returns the grandparent (Level 2) node object of a page node.

Syntax

```
Set <pNode> = <p>.Tree.L2Parent
```

The L2Parent property syntax has the following parts:

Table 8-61: L2Parent Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_0 containing the record structure of level 0 (page)
pNode	A return value of type IAS_RECORD_2, the grandparent (level 2) node object of a page node.

Comments

Use the L2Parent property to get the grandparent node of a page node. The L2Parent property is read-only.

This property is only available for level 0 nodes. For level 1 nodes, use the Parent property. Other level nodes do not have an L2Parent.

Example

This example reports the node ID of the level 2 parent node to the application Event Log:

```
Option Explicit

Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim pL2Parent As IAS_RECORD_2
    Set pL2Parent = p.Tree.L2Parent
    Call IALogNTMessage(IALOG_INFO, "L2 Parent has NodeID of " & pL2Parent.Tree.NodeID)
End Sub
```

L3Parent Property

The L3Parent property returns the node object of the level 3 ancestor node.

Syntax

```
Set <pNode> = <p>.Tree.L3Parent
```

The L3Parent property syntax has the following parts:

Table 8-62: L3Parent Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_*<n>* containing the record structure of level *<n>*, where *<n>* is 0 (page) or 1 (document).
pNode	A return value of type IAS_RECORD_3, the grandparent (level 3) node of a page node.

Comments

Use the L3Parent property to get the great-grandparent node of a page node or the grandparent node of a level 1 node. The L3Parent property is read-only.

This property is only available for level 0 and level 1 nodes. For level 2 nodes, use the Parent property. Other level nodes do not have an L3Parent.

Example

This example reports the node ID of the level 3 parent node to the application Event Log.

```
Option Explicit

Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim pL3Parent As IAS_RECORD_3
    Set pL3Parent = p.Tree.L3Parent
    Call IALogNTMessage(IALOG_INFO, "L3 Parent has NodeID of " & pL3Parent.Tree.NodeID)
End Sub
```

NextInLevel Property

Returns the node object of the next same-level (cousin) node.

Syntax

```
Set <pNode> = <p>.Tree.NextInLevel
```

The NextInLevel property syntax has the following parts:

Table 8-63: NextInLevel Property Syntax

Part	Description
p	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where n is an integer from 0 (page) through 6.
pNode	A return value of type `IAS_RECORD_<n>`, the node object of the next same-level node.

Comments

The NextInLevel property returns the next cousin node (the next node at the same level, regardless of whether it has a different parent.) If there is no next node, then *pNode* is set to `Nothing`. This property is not available for the root (level 7) node, because only one root node exists per batch. The NextInLevel property is read-only.

Example

This example loops through all of the level 2 nodes in the batch. A dynamic value called `MyVal` is created for each level 2 note and stores the string “Hi there”.

```
Option Explicit

Private Sub ImageExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim pL2Node As IASLib.IAS_RECORD_2
    Set pL2Node = pRoot.Tree.L2Child(0)
    While Not pL2Node Is Nothing
        pL2Node.ImageExp("MyVal") = "Hi there"
        Set pL2Node = pL2Node.Tree.NextInLevel
    Wend
End Sub
```

NextPeer Property

Returns the node object of the next peer (sibling) node.

Syntax

```
Set <pPeer> = <p>.Tree.NextPeer
```

The NextPeer property syntax has the following parts:

Table 8-64: NextPeer Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_*<n>* containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 6.
pPeer	A return value of type IAS_RECORD_*<n>*, the node object of the next peer (sibling) node, where <*n*> is the same as the level at which the NextPeer property executes.

Comments

The NextPeer property returns the next peer node (the next node at the same level with the same parent.) If there is no next peer, then *pPeer* is set to Nothing. This property is not available for the root (level 7) node, because only one root node exists per batch. The NextPeer property is read-only.

Example

This example loops through all of the level 2 nodes within the first level 3 node. It creates the dynamic value called MyVal (which stores the string “Hi there”) for each level 2 node.

```
Option Explicit
Private Sub ImageExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim pL2Node As IASLib.IAS_RECORD_2
    Set pL2Node = pRoot.Tree.L2Child(0)
    While Not pL2Node Is Nothing
        pL2Node.ImageExp("MyVal") = "Hi there"
```

```
        Set pL2Node = pL2Node.Tree.NextPeer
    Wend
End Sub
```

Node Property

Returns the ID of the current node. This property is available for backwards compatibility only. Use instead the new NodeID property.

Syntax

```
<lNodeID> = <p>.Tree.Node
```

The Node property syntax has the following parts:

Table 8-65: Node Property Syntax

Part	Description
p	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch).
`lNodeID`	A return value of type Long, the ID of the current node.

Comments

The Node property is identical to the NodeID property. NodeID was added when NodeCount property was added with the aim of differentiating multiple node-related properties. Use the NodeID property because its name is more descriptive, and because its future compatibility is guaranteed. The Node property is read-only.

NodeCount Property

The NodeCount property returns the index of a current node within a specified higher level.

Syntax

```
<lIndex> = <p>.Tree.NodeCount(<lLevel>)
```

The NodeCount property syntax has the following parts:

Table 8-66: NodeCount Property Syntax

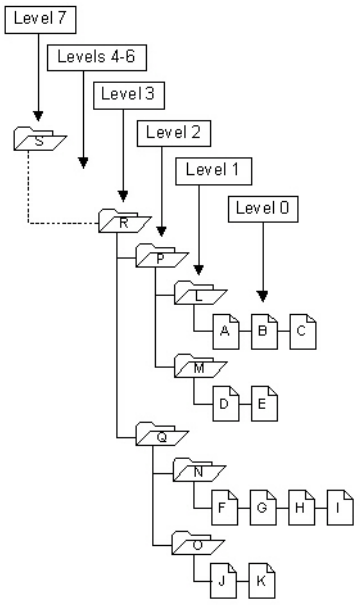
Part	Description
p	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where n is an integer from 0 (page) through 6.

Part	Description
`lLevel`	A variable of type Long, the level at which to count nodes. Must be at least one level higher than the level specified by p.
`lIndex`	A return value of type Long, the index of the current node. (0-based.)

Comments

The NodeCount property returns a number indicating the position of the current node among all same-level nodes within the level specified by *`lLevel`*. The position of a node is determined by counting each consecutive same-level node within the specified level, starting with 0. The NodeCount property is read-only.

As an example, assume that you are working with the batch depicted in the following diagram and are currently processing node `H`. The `lLevel` argument of the NodeCount property is set to 1. In this case, NodeCount returns 2 because node `H` is the third node within its level 1 parent. (Remember that the *`lLevel`* argument is zero-based.)



If you change the *`lLevel`* argument to 3, then NodeCount return 7, because node `H` is the eighth same-level node within its level 3 ancestor.

This property is not available for the root (level 7) node, because a batch can have only one root node.

Example

This example reports to the application event log the `NodeCount` for each node greater than level 1:

```
Option Explicit

Private Sub ScanExp_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim i As Integer

    For i = 1 To 7
        Call IALogNTMessage(IALOG_INFO, "NodeCount(" & i & ")=" & p.Tree.NodeCount(i))
    Next i
End Sub
```

NodeID Property

Returns the ID of the current node.

Syntax

```
<lNodeID> = <p>.Tree.NodeID
```

The NodeID property syntax has the following parts:

Table 8-67: NodeID Property Syntax

Part	Description
p	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 7 (batch).
`lNodeID`	A return value of type Long, the ID of the current node.

Comments

The NodeID property retrieves the ID from the current node. It is useful when constructing keys for IAValueGet/Set methods, in which you often use a node ID within a key string. The NodeID property is read-only.

Example

This example reports to the application event log that the `Finish` event handler has been called for the current node:

```
Option Explicit

Private Sub ImageExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Call IALogNTMessage(IALOG_INFO, "Finished Node " & pRoot.Tree.NodeID)
End Sub
```

Related Topics

"IAValueGet/Set Methods" on page 576

NumChildren Property

Returns the number of children of the specified level.

Syntax

```
<lChildren> = <p>.Tree.NumChildren(<lLevel>)
```

The NumChildren property syntax has the following parts:

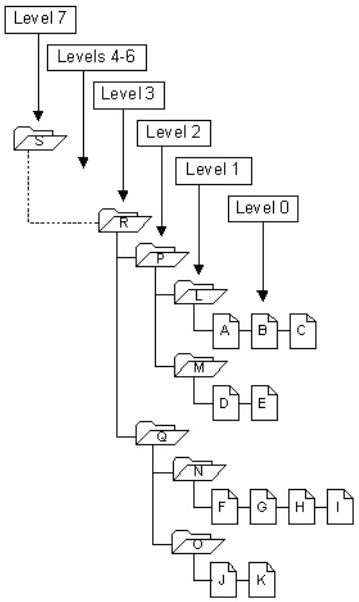
Table 8-68: NumChildren Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_<*n*> containing the record structure of level <*n*>, where <*n*> is an integer from 1 (document) through 7 (batch).
lLevel	A variable of type Long, the level at which to count children. Must be at least one level lower than the level specified by p.
lChildren	A return value of type Long, the number of child nodes.

Comments

The NumChildren property gets the number of direct descendents that belong to a specified level. The NumChildren property is read-only.

For step, assume that you are working with the batch depicted in the following diagram and are currently processing node R. The *lLevel* argument of the NumChildren property is set to 2. In this case, NumChildren returns 2 because



nodes P and Q are the level 2 children of node R.

If you change the *lLevel* argument to 0, then NumChildren returns 11, because node R has 11 children (nodes A through K) at page level.

This property is not available for level 0 nodes, because they have no children.

Example

This example counts the number of level 2 children within the batch such as, the level 7 task node. It then loops through each level 2 child.

```
Option Explicit

Private Sub IndexExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim Index As Long     Dim Count As Long
    Dim pL2Child As IASLib.IAS_RECORD_2
    Count = pRoot.Tree.NumChildren(2)

    For Index = 0 To Count - 1
        Set pL2Child = pRoot.Tree.L2Child(Index)
        'Add your code here which performs some function on
        'each pL2Child
    Next
End Sub
```

Page Property

Returns the node object of the page at the specified index.

Syntax

```
Set <pNode> = <p>.Tree.Page(<lIndex>)
```

The Page property syntax has the following parts:

Table 8-69: Page Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_<*n*> containing the record structure of level <*n*>, where <*n*> is an integer from 1 (document) through 7 (batch).
lIndex	A variable of type Long, the index of the page whose Node ID you want to get. (0-based.)
pNode	A return value of type IAS_RECORD_0, the node object of the page whose index you specified.

Comments

The Page property returns the page-level node of the specified index number. *lIndex* is zero-based, so legal index values range from 0 to (number of children - 1). The Page property can only get page nodes that are direct descendents of *p*. The Page property is read-only.

Use this property if you want to access page nodes from a higher level without first navigating to the page level. This property is not available for Level 0 Nodes, because they have no children.

Example

This example loops through all of the level 0 (page) nodes of the task node. It creates a dynamic value named `NewVal` (which stores the value 12) for each page node:

```
Option Explicit
Private Sub IndexExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim Index As Long
    Dim Count As Long
    Dim pPage As IASLib.IAS_RECORD_0
    Count = pRoot.Tree.NumChildren(0)

    For Index = 0 To Count - 1
        Set pPage = pRoot.Tree.Page(Index)
        pPage.IndexExp("NewVal") = "12"
    Next
End Sub
```

Pages Property

Returns a collection of page nodes, which can be accessed in a `For Each` loop

Syntax

```
For
Each <pPage> in <p>.Tree.Pages
```

The Pages property syntax has the following parts:

Table 8-70: Pages Property Syntax

Part	Description
`p`	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where <*n*> is an integer from 1 (document) through 7 (batch).
`pPage`	An objects of type `IAS_RECORD_0`, returned for each page in the `For Each` loop.

Comments

The Pages property returns a collection object, which is used to loop through all of the page nodes of the object. Compared to using the Page property, the Pages property usually simplifies the code required when performing operations on each page node. The Pages property is read-only and can exist only for level 1 and higher nodes.

Example

This example shows how to loop through each level 0 (page) node. It uses the Pages property to create a dynamic value named `NewVal` (which stores the value 12) for

each page node. Compare this example to the example using the Page property to view how the Pages property can be easier to use.

```
Option Explicit

Private Sub IndexExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim pPage As IASLib.IAS_RECORD_0

    For Each pPage in p.Tree.Pages
        pPage.IndexExp("NewVal") = "12"
    Next
End Sub
```

Related Topics

"Page Property" on page 602

Parent Property

Returns the specified parent node object.

Syntax

```
Set <pNode> = <p>.Tree.Parent
```

The Parent property syntax has the following parts:

Table 8-71: Parent Property Syntax

Part	Description
p	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level <*n*>, where <*n*> is an integer from 0 (Page) through 6.
pNode	A return value of type `IAS_RECORD_<m>`, the node object of the parent node, where <*m*> is one greater than the level at which the `Parent` property executes.

Comments

The Parent property gets the node object exactly one level higher than the level specified by <*p*>. The Parent property is read-only

This property is not available for the root level (level 7) node, because it has no parents.

Example

This example reports the parent node ID of the current task to the application Event Log:

```
Option Explicit
Private Sub ScanPlus_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim pParent As IASLib.IAS_RECORD_1

    Set pParent = p.Tree.Parent
    Call IALogNTMessage(IALOG_INFO, "Parent has NodeID of " & pParent.Tree.NodeID)
End Sub
```

PrevInLevel Property

Returns the node object of the previous same-level (cousin) node.

Syntax

```
Set <pNode> = <p>.Tree.PrevInLevel
```

The PrevInLevel property syntax has the following parts:

Table 8-72: PrevInLevel Property Syntax

Part	Description
p	An object of type `IASLib.IAS_RECORD_`*<n>* containing the record structure of level <*n*>, where <*n*> is an integer from 0 (page) through 6.
pNode	A return value of type `IAS_RECORD_`*<n>*, the node object of the previous same-level node, where <*n*> is the same as the level at which the `PrevInLevel` property executes.

Comments

The PrevInLevel property returns the previous cousin node (the previous node at the same level, regardless of whether it has a different parent.) If there is no previous node, then *pNode* is set to `Nothing`. This property is not available for the root (level 7) node, because only one root node exists per batch. The PrevInLevel property is read-only.

Example

This example loops through all of the level 2 nodes in the batch in reverse order. It creates a dynamic value called `MyVal` (which stores the string “Hi there”) for each level 2 node.

```
Option Explicit

Private Sub ImageExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim pL2Node As IASLib.IAS_RECORD_2
    Set pL2Node = pRoot.Tree.L2Child(pRoot.Tree.NumChildren(2) - 1)
    While Not pL2Node Is Nothing
        pL2Node.ImageExp("MyVal") = "Hi there"
        Set pL2Node = pL2Node.Tree.PrevInLevel
```

```
    Wend
End Sub
```

PrevPeer Property

Returns the node object of the previous peer (sibling) node.

Syntax

```
Set <pPeer> = <p>.Tree.PrevPeer
```

The PrevPeer property syntax has the following parts:

Table 8-73: PrevPeer Property Syntax

Part	Description
p	An object of type `IASLib.IAS_RECORD_<n>` containing the record structure of level n, where n is an integer from 0 (page) through 6.
pNode	A return value of type `IAS_RECORD_<n>`, the node object of the previous peer (sibling) node, where <*n*> is the same as the level at which the PrevPeer property executes.

Comments

The PrevPeer property returns the previous peer node (the previous node at the same level with the same parent). If there is no previous peer, then *`pPeer`* is set to `Nothing`. This property is not available for the root (level 7) node, because only one root node exists per batch. The PrevPeer property is read-only.

Example

This example loops through all of the level 2 nodes in the last level 3 node in reverse order. It creates a dynamic value called `MyVal` (which stores the string “Hi there”) for each level 2 node.

```
Option Explicit

Private Sub ImageExp_Finish(ByVal pRoot As IASLib.IAS_RECORD_7)
    Dim pL2Node As IASLib.IAS_RECORD_2
    Dim Num As Long

    Num = p.Tree.NumChildren(2)
    Set pL2Node = pRoot.Tree.L2Child(Num - 1)
    While Not pL2Node Is Nothing
        pL2Node.ImageExp("MyVal") = "Hi there"
        Set pL2Node = pL2Node.Tree.PrevPeer
    Wend
End Sub
```

Root Property

Returns the root (level 7) node object.

Syntax

```
Set <pRoot> = <p>.Tree.Root
```

The Root property syntax has the following parts:

Table 8-74: Root Property Syntax

Part	Description
p	An object of type IASLib.IAS_RECORD_<*n*> containing the record structure of Level <*n*>, where <*n*> is an integer from 0 (Page) through 6.
pRoot	A return value of type IAS_RECORD_7, the root node object of the batch.

Comments

The Root property retrieves the root node object, providing a convenient way to navigate to the root node quickly. The Root property is read-only.

This property is not available for the root (level 7) node.

Example

This example reports the node ID of the root node to the application Event Log. In addition, this example displays how to retrieve the root node object while working within a record structure at a level other than level 7.

```


Option Explicit

Private Sub ImageExp_Finish(ByVal p As IASLib.IAS_RECORD_0)
    Dim pRoot As IASLib.IAS_RECORD_7

    Set pRoot = p.Tree.Root
    Call IALogNTMessage(IALOG_INFO, "Root has NodeID of " & pRoot.Tree.NodeID)
End Sub
```

Events

This section explains the syntax of event handlers.

Glossary

ACL

Access Control List

API

Application Programming Interface

ASCII

American Standard Code for Information Interchange

BDF

FormWare Batch Definition File

BOF

Business Objects Framework

CAR

Courtesy Amount Recognition

COM

Component Object Model

CPU

Central processing unit

CS

Content Server

DFL

Dispatcher Function Library

DIA

Dispatcher for InputAccel

DLL

Dynamic Link Library

DOM

Dispatcher Object Model

GAC

Global Assembly Cache

GUI

Graphical User Interface

HPA

High Precision Anchor

IAP

InputAccel process file extension

IAS

InputAccel Server

IDE

Integrated Development Environment

IPP

Integrated ProcessFlow Project

JDF

Job Definition File

LAR

Legal amount recognition

LIFFE

Line Item Free Form Engine

MDF

Module Definition File

MSDN

Microsoft Developer Network

OCR

Optical Character Recognition

ODBC

Open Database Connectivity

OLE

Object Linking and Embedding

PAL

Production Auto-Learning

PCF

Process Configuration File

PDA

Calera Processed Document Architecture file extension

SBL

Softbridge Basic Language

SBO

Service-Based Object

SOAP

Service-Oriented Access Protocol

TIFF

Tagged Image File Format

URI

Uniform Resource Identifier

URL

Uniform Resource Locator

US

United States

UTF-8

Unicode Transformation Format 8–bit

VBA

Microsoft Visual Basic for Applications

WSDL

Web Services Description Language

XML

Extensible Markup Language