opentext™

OpenText™ Intelligent Capture

Recognition and Advanced Recognition Modules Guide

This guide contains information about the Intelligent Capture recognition and advanced recognition client modules.

ECPCORE220300-CMR-EN-1

OpenText™ Intelligent Capture
Recognition and Advanced Recognition Modules Guide
ECPCORE220300-CMR-EN-1
Rev.: 2022-June-20

This documentation has been created for OpenText™ Intelligent Capture CE 22.3.
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

Recognition and Advanced Recognition Modules

Starting in release 22.3, this new guide contains a compilation of the Intelligent Capture recognition and advanced recognition client modules. In previous releases, these were available as separate guides.

For information about common features and functionalities in the Intelligent Capture client modules, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

Table 1-1: Recognition Modules

Module	Original module ID	Link
Extraction	ecpcore-cex	See "Extraction"
NuanceOCR	ecpcore-cnu	See "NuanceOCR"
Standard OCR	ecpcore-cob	See "Standard OCR"

Table 1-2: Advanced Recognition (AR) Modules

Module	Original module ID	Link
Classification	ecpcore-ccf	See "Classification Module"
Collector	ecpcore-cco	See "Collector Module"

Chapter 2

Extraction

This section includes the Intelligent Capture Extraction module: **ecpcore-cex**.

2.1 Introduction

This guide describes the extraction module. Extraction is an optional step in a process. You can use this Intelligent Capture client module to automatically extract data from documents using optical character recognition (*OCR*).

At run time, the Extraction module identifies the printed or handwritten characters that make up a document and returns the information in the form of text. Extraction is triggered at any level (0 through 7).

For more information see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

2.1.1 Relationship with Recognition Projects, Document Types, and Templates

During CaptureFlow design, you can set up the required components for the extraction step. Begin by creating a *recognition project* for a paper form in Recognition Designer. A recognition project identifies the templates, base images, and rules for classifying documents. It also identifies the *OCR* engine(s) to use for extracting data from documents. You can use a single recognition project for all document capture jobs or create different projects based on specific processing needs.

Next you create a *document type* for the paper form using the Document Designer and associate it with the recognition project. The document type defines the data entry form that Completion module operators use for indexing and validation. Document type definition includes designing the layout of the data entry form and specifying document-level validation rules.

When you save the new document type, Document Type Editor generates an *indexing family* within the recognition project. The indexing family contains all the index fields for all pages of the document.

After document type creation, use the Recognition Designer to create *page templates* for the document type and then map them to the indexing family. A page template identifies an image. It includes the following:

- A label for the template.
- Information that identifies the template for classification purposes. For example, an IBM logo and text such as **Total Amount Due** can identify a page image as an IBM invoice.

- Layout information that specifies where data is located on the form (zones).
- Configuration information that specifies how to extract the data, including which OCR engines to use and how to pre-process the image.

Each page of the document can be associated with a different page template.

After configuring page templates for a form, place index family fields on the template and configure the extraction settings in Recognition Designer. Fields can also be placed on the template automatically using the Template Wizard. This information, which is saved in the recognition project, identifies which data should be extracted from the image and written to the data entry field.

For example, assume you have an index family called **Telephone Bill**. You can position the **Total Amount Due** field on the bill on the **Telephone Bill Page 1** template, the **Usage Charges** field on page 2 on the **Telephone Bill Page 2** template, and so on.

A field can be positioned on multiple pages in the document. For example, the **Total Amount Due** field might appear on both the first and last page. In this case, the data from the first extracted page is written to the field.

Prior to Extraction module setup, ensure that the location of the recognition projects folder is defined on the **System Configuration** tab of the **System** area in Intelligent Capture Designer. The Extraction module uses the specified path to locate your recognition projects. Then during Extraction module setup, select the recognition project to be used for the step. This setting, along with other setup options, determines how the Extraction module recognizes pages when it processes tasks in production mode.

2.1.2 Process Flow

The process flow for the Extraction module at run time is as follows:

1. The document is pre-separated, for example, using Classification, Identification, or some other intelligent document assembly mechanism.
2. Extraction sets the internal document type to construct the internal document type data object to the first of:
   - The document-level <*UimDocumentType*> if non-blank and valid (that is, found in the list of document types). This case is used when the structured documents are classified by setting the document type directly (such as through a barcode) rather than by using Classification.
   - The document type associated with the template of the first page.
   - If the above methods fail, the document does not have a document type. However, the task does not fail.
3. Extraction chooses the template for each page as follows:

- Extraction determines the page template from the <*PageTemplateId*> value. This value either skips the page from data extraction, or contains a valid template ID which is taken, or tells Extraction to proceed to input XML data.
- If specified by the <*PageTemplateId*> value, Extraction parses input XML data (page-level <*InputPageDataXml*> value). Input *XML* must be set to the output of Classification or Identification. If the XML or the page template ID that it contains is invalid, nothing is extracted from that page.
- If <*InputPageDataXml*> is blank, Extraction determines the page template from the document type (as determined by the steps in the previous bullet) as follows:
  - At design time, it assigns the ordered set of template IDs using Recognition Designer. This set of template IDs is stored as part of the document type definition.
  - The first page in the document is assigned the first template ID, the second page is assigned the second template, and so on. For example, if a document type has template IDs 10, 11, 12, and 13 and <*InputPageDataXml*> does not exist for the third page in the document, Extraction would use template ID 12 (the third item in the list) regardless of whether the first two pages had <*InputPageDataXml*>.
  - If the number of document pages exceeds the number of templates, the extra pages are not associated with a template and therefore are not recognized. If there are more templates than pages, the extra templates are not used.
- If none of the above (<*PageTemplateId*>, <*InputPageDataXml*>, or a document type indicated by <*UimDocumentType*>) includes a valid page template, the page is not associated with a template and no data is extracted from the page. However, the task does not fail.

4. The Extraction module extracts data from each page using *OCR*.
5. The Extraction module reconciles data from the extracted pages into a document-level object.

   For more information, see "Inputs and Outputs" on page 12.
6. The data is then validated against what is defined at design time:
   - Field validation (required fields, masking, and so on).
   - Document-level validation rules, both declarative and scripted.
   - Validations at a higher level than document, such as verifying that a field that should be common across a set of all documents, are handled by Completion rather than Extraction.
7. The Extraction module generates <*OutputPageDataXml*>, which is bound to the Completion module's <*PageDataXml*> in a typical workflow process.

2.1.3 Inputs and Outputs

The Extraction module uses the following as input:

- *Page images* for a document. The document must have been pre-separated in some way using Classification, Identification, or some other intelligent document assembly mechanism.

  Note: You can restrict extraction to specific pages. For more information, see the **Pages to Process** option in "Setting Up Extraction" on page 16 and the <*PagesToProcess*> IA value in "IA Values" on page 23.

- *Document type*. At run time, the document type can be determined in the following ways:

  – By classification if the optional Classification module is used. In this case, the page template for the first page of the document identifies the document type for the document.

  – By manual separation and barcode. In this case, manual separation provides boundaries for the document and the barcode sets the IA value for the document type. When the document type is set using an IA value and the page template is left blank, the document type also determines the page template that is used. Templates are assigned to document pages based on the alphanumeric order of the template name as defined in Recognition Designer. For example, templates named **1**, **2**, and **10** would be assigned to pages as follows: Page 1 to **1**, Page 2 to **10**, and Page 3 to **2**.

  – By Completion module operators if the ability to change or assign a document type during production is enabled during Completion setup. Assuming that your business process requires documents with changed types to be routed back to Extraction (to extract data for the new document type), you must map the newly assigned document type from the Completion output to the input document type for the Extraction step. This procedure is necessary because the Extraction step does not use a document type object as input. All steps in the process that follow the second Extraction step should use only the new output data from Extraction.

- *Page template(s)* associated with the indexing family (an internal representation of the document type in the recognition project), and field placements that identify which data should be extracted from the paper image and written to the data entry fields. If a page template is set, the document type is determined by the first page.

The Extraction module extracts field data from the document on a page-by-page basis using the field placements found for each page template. Field data is extracted as follows:

- Single values for the same field (for example, the same invoice number on multiple pages) are reconciled into one value as follows:

  - If the per-page value is blank on all pages, the document type field is blank.
  - If the per-page value is the same on all pages that have a non-blank value, that is the value for the document type field.
  - For primary (non-array) fields, either the first or last found field (as set by **Extract Page** property in the document type definition) determines the value assigned to the field. In other words, if the same field appears on multiple images in the document, the data from the first or last extracted page is written to the field. In addition, if the values differ from page to page, then the field is flagged for manual confirmation independent of the confirmation setting in the document type. For example, if the **Total Amount Due** field is found on both the first and last page and **Extract Page** is set to first, then the value from the first page is used.

- For array fields, the row data is concatenated. For example, if a table in an **Invoice** document type that contains a list of purchase items is split across two pages, the row data is concatenated together to produce the full table.
- Table data is constructed by populating the document type data with only the extracted values where the line type is primary. Secondary lines may be processed in scripting but are dropped by Extraction.

The Extraction module then generates a document type object containing the extracted field data in the form of IA values, including:

- Field data: Data from fields defined by the page template(s).
- Table data: The name and zone coordinates of the table and the values of all the cells in the table.

The document type object produced by the Extraction module serves as input to module steps later in the process such as Completion or Standard Export. Completion uses this object to identify the document type and index fields for the document. It then retrieves the index values from the document type object and pre-populates the data entry form using data from the recognized pages. Operators can then verify the accuracy of the extracted data.

2.1.4 Relationship with Scripting

You can use profile scripting to manipulate the document data or data entry form. For example, you may need to validate a certain extracted field value against other document values or external data. Or, if extraction of a particular field was not successful, you may need to calculate that value or retrieve it from an external source and store the result in the document data structure.

The order of execution for scripting events in Extraction is:

1. *<DocumentExtracted>*: Executed after all extracted values have been copied into the document type data.
2. *<ExecuteValidationRule>*, *<ExecuteTableRowValidationRule>*: Executed once per validation rule.

3. *<DocumentUnload>*: Executed just before the task is completed; if document type data is changed in this event, the rules that depend on the affected fields are not rerun.

For detailed information, see *OpenText Intelligent Capture - Scripting Guide (ECPCORE-PSC)*.

2.1.5 Document Type Statistics

Both the Extraction and Completion modules provide statistics. These statistics, which pertain to production data, can help you identify issues that affect productivity. You can:

- Measure operator productivity and performance tuning.
- Find classification problems: Identify pages for which classification failed or documents for which the document type was changed.
- Find extraction problems: Identify fields that contained low-confidence characters or for which the value was changed after extraction.
- Find throughput bottlenecks: Identify steps where extraction or validation were slow.

You specify whether statistical data is collected as part of the step configuration settings in each process. The Extraction module provides Setting Up Extraction. Completion provides options for collecting document and field data.

2.1.5.1 Where Data is Stored

Statistical data is stored in the following tables:

- **Template**: Stores information about the pages processed by the Extraction module. You can use this data to improve extraction results for a particular page template. Information is divided by process to enable you to organize your data by line of business in cases where templates are shared.
- **Document Type**: Stores information about the documents processed by the Extraction module and Completion. You can use this data to determine which document types take the longest time to process or have high reclassification rates. Information is divided by process to enable you to organize your data by line of business in cases where templates are shared. Capturing character counts enables accuracy measurement by calculating the percent of characters that were changed.
- **Field**: Stores information about the fields changed in Completion; records are created only for fields that the operator changes. You can use this data to determine which fields take the longest time to process or have high data correction rates. Capturing character counts enables accuracy measurement by calculating the percent of characters that were changed.

Each table contains three tags (*<ReportTag1>*, *<ReportTag2>*, and *<ReportTag3>*) that can be used to further divide the statistical data according to business needs. You

can use these tags to capture separate data in cases where capture services are shared across an organization. For example, you could organize the statistics by functional area by setting <*ReportTag1*> to the functional area for each task, such as `Accounting` for one task and `Sales` for a second task. Another example might be to organize by geographical region such as `North America` versus `Europe`. You could group the statistics by these tags to show operator productivity for each functional area or geographical region.

Note: The string values you specify for these tags are written to the database exactly as entered. No processing is performed on the values. The tags are set by the IA process as task-level values with the same name.

Use these tags only when you need to further break down statistical data. Capturing additional, unused information increases the number of records in the database unnecessarily.

For a description the report tables, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*

2.1.5.2 How Data is Captured

Each running module creates new records in the three reports tables (`Template`, `Document Type`, and `Field`) as needed, but no less frequently than one per hour. The cumulative statistics for a given date and hour are computed by aggregating all rows for that time. For example, the number of pages processed by Extraction on 1/23/2012 between 10:00 and 11:00 is the total of all records where the `Date` is set to `1/23/2012 10:00`.

2.2 Setting Up Extraction

Most configuration information for Intelligent Capture client modules is defined during *module setup*. You can run a module in setup mode from Intelligent Capture Designer, from Intelligent Capture Administrator, or from a command prompt with appropriate arguments. For more information, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

When launched in setup mode, a client module displays the **Setup** window in which you can specify the module's configuration settings.

2.2.1 Understanding Data Flattening

The Extraction module stores document data in a single *UimData* IA value that contains all fields in the document. Extraction can output document data to the next CaptureFlow step in the original UIM data structure, or you can choose to convert document data into a set of dynamic values, each holding a single extracted field, in addition to outputting the data in its original structure.

Note: This option is enabled only when setup is launched from Intelligent Capture Designer. It cannot be configured if setup is run from Intelligent Capture Administrator.

The process of converting UIM data into "flat" IA values is referred to as *data flattening*. During setup, you can choose to flatten the extracted document data or to output it in its original structure. When making your choice, consider the following:

- In a typical CaptureFlow, data extraction is followed by validation or export. If you use the Completion module for data validation, you can output the *UimData* IA value as is. Completion supports UIM data and can flatten document data before output if necessary.
- If the next step is Standard Export then you do not need to flatten document data either. However, pre-7.0 exporters (for example, Documentum Advanced Export) cannot read data from UIM structures, and document data flattening is required.
- Flattening is required if the extracted document data is used in a CaptureFlow for making decisions, such as choosing the next step. CaptureFlow Designercannot read data from UIM structures.
- Flattening index data can increase batch size, affecting performance.

If you need individual values, you must configure them using the Setting Up Extraction for IA values. During step configuration, you can choose to create new values or to only populate existing ones. You can also specify the step destination into which the values should be written. For example, you can use Defining a custom value for a CaptureFlow in CaptureFlow Designer when creating the CaptureFlow, which are then populated by the module. Another option is to use a separate data-only *MDF* that is populated by the module.

2.2.2 Setting Up Extraction

Configure options on the **Setup** pane in module setup to specify how the Extraction module should recognize pages when it processes tasks in production mode.

To set up Extraction:

1. Run the Extraction module for setup.

   Note: If you need to output the Understanding Data Flattening document data, launch setup from Intelligent Capture Designer. Data flattening cannot be configured if setup is run from Intelligent Capture Administrator.

2. Specify the required settings as described in the table:

Table 2-1: Extraction module: Setup Settings

Setting		Description
Recognition Project		Expand the drop-down list and select the recognition project to use for this step. The list contains all projects found under the path defined by the **RecognitionProjectSharedDir** property in Intelligent Capture Designer.
Pages to Process	**First Pages**	Starting from the first page, specify the number of pages on which to perform extraction. **Note:** You can also use the <*PagesToProcess*> IA value. For more information, see "IA Values" on page 23.
	Last Pages	Starting from the last page and counting backwards, specify the number of pages on which to perform extraction.

Setting		Description
Output IA Values	**Destination**	Expand the drop-down list and select the destination and format for outputting the document data. Values: • **(Do not write to IA values)**: Select this option to output the document data in its original structure. When selected, the rest settings in the **Output IA Values** group are disabled. • **<step name>**: Select the step to which you need to output document data after Understanding Data Flattening. • **Custom Values**: Select this option to output flattened document data into custom values available in your CaptureFlow.

<table>
<tr><th colspan="2">Setting</th><th>Description</th></tr>
<tr><td></td><td>Output Array Fields</td><td>If data flattening is enabled in the Destination setting, specify flattening for array data (extracted table data). Values:
Value Per Row: Output each array item as a separate IA value with the <FieldName>_<row #> name where the row number starts at 1.
Value Per Array Field: Output each table field (all rows) as a single <CR><LF> delimited string value with the <FieldName> name. The last item in the string is not followed by the delimiter.
In both cases, the number of table records is output to IA value <TableName>_count.
The Fruits table includes the following fields and content:<table><tr><th>ID</th><th>Description</th></tr><tr><td>1</td><td>Orange</td></tr><tr><td>2</td><td>Apple</td></tr></table>If you select the Value Per Row option, the table content will be output as follows:
Id_1=“1”
Description_1=“Orange”
Id_2=“2”
Description_2=“Apple”
Fruits_count=“2”
If you select the Value Per Array Field option, the</td></tr>
</table>

Setting		Description
		table content will be output as follows: • `Id`="1<CR><LF>2" • `Description`="Apple<CR><LF>Orange" **Note:** This value can be mapped to repeating attributes when usingDocumentum Advanced Export. • `Fruits_count`="2"
	Output Level	Expand the drop-down list and select the node to output the flattened document data: • **Document Only**: set by default. Select to output data at document level. • **Field Index Level Only**: select to output data in accordance with the **Index Level** specified in Intelligent Capture Designer for each field in a document type. • **Document And Field Index Level**: select to use both approaches to output data. **Note:** When outputting to the index level, the Extraction module ignores empty output values so that the preceding outputs are not overwritten with blanks.

Setting		Description
	Always import higher level values	This setting lets you propagate data between tasks. Disabled if the **Document Only** output level is selected. Check this option to auto-populate document fields with the common output values if any exist in the level higher than the task level. When selected, after recognition is completed, data is propagated in accordance with the selected output destination and at the index level specified for this field in the document type. When using **CustomValues** for data import, consider the UimDataImportMode allowed values.

Setting		Description
	Output As Dynamic Values	Check this option to create a dynamic value for each field that does not have a matching IA value defined in the selected destination (such as an MDF value or a custom value). Example: If a document type has `Name` and `Date` fields but the output destination has only a `Name` IA value, then selecting this option would populate the `Name` value and create a dynamic `Date` value. If you choose not to use dynamic values, the `Name` field's value would be written to the existing destination IA value and the `Date` field value would be skipped. Using dynamic values is also necessary if arrays are flattened with the **Value Per Array Field** option (where all values cannot be pre-created because the number of items in the array is not known in advance).
Statistics	**Collect template statistics**	Check this option to log statistics about the processed templates.
	Collect document type statistics	Check this option to log statistics about the processed documents.

3. Click **OK** to save the configuration settings.

2.3 Running Extraction in Production

After the module has been set up, tested, and placed in a production environment, operators and administrators will run the module in production.

For information related to common production tasks for unattended modules, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*. For information on production topics that are common to unattended modules and may not apply to this module, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

Note: You can use the Advanced Cloud OCR engine only when you enable extraction with information extraction-based projects. To use the Advanced Cloud OCR engine, you require a valid credential file from Google with the following content:

```
{ "type": "", "project_id": "", "private_key_id": "", "private_key": "",
"client_email": "", "client_id": "", "auth_uri": "", "token_uri": "",
"auth_provider_x509_cert_url": "", "client_x509_cert_url": "" }
```

The plugin uses the environment variable *<GOOGLE_APPLICATION_CREDENTIALS>* to get the credentials file. If the environment variable is not set or is empty, the plugin uses the `GoogleConfig.json` file located in the `IEE2\Config` folder.

2.4 Reference—Extraction

The topics within this section contain reference information useful while using the application in setup or production.

2.4.1 IA Values

Intelligent Capture provides both common and module-specific IA values. Common IA values, which are described in the *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)* , are valid for all modules.

IA values that can be used with the Extraction module are described in the following table. The **Type** column identifies the required data type (file, string, and so on) for each IA value followed by the type of IA value (input and/or output):

- Input IA values include input file variables, which serve as pointers to stage files, and input processing variables, which store setup data that a module uses to process tasks, including default and user-defined module settings.
- Output IA values include output processing variables, which store data that was generated by the module during processing. Some of the following output values are created dynamically and do not need to be specified in the `MDF` file.

Table 2-2: IA Values for Extraction

IA Value	Level	Type	Description
<DispatcherProjectName>	T	String, Input	A task level value that holds the name of the recognition project (DPP file) to be used by default. This value should only contain the DPP name and not the path. If the value is empty, then the DPP specified in Extraction setup is used. If the value contains a DPP which does not exist, an error message is shown.

IA Value	Level	Type	Description
<*DocStatus*>	1	DocumentStatus, Output	Object describing the state of each document, which can be used for routing decisions in the workflow. The document status is needed following a Classification, Recognition, or Validation step. The value is created when the task is closed, independent of whether it is completed or canceled, and is not automatically updated. The document status is made up of the following properties (and types), per document: • <*CharactersInvalid*> (Long): Number of remaining questionable characters. This is initially set during recognition, but a given operator may not finish all of them. If this value is nonzero, the document contains work that can be corrected by a step configured for Character work level or higher. • <*FieldsInvalid*> (Long): Number of fields that failed field-level validation, either due to built-in validation such as data type, range,

IA Value	Level	Type	Description
			or required properties; or due to a validation rule with only a single associated field. This number is zero if there are no single-field validation errors in the document. If this number is nonzero, it contains work that can be corrected by a step configured for Field work level or higher. • <*FieldsUnconfirmed*> (Long): Number of unconfirmed fields. • <*FieldsFlagged*> (Long): Number of fields that are flagged. • <*PagesFlagged*> (Long): Number of pages that are flagged. • <*TotalFlagged*> (Long): Number of fields and pages that are flagged, plus one if the document is also flagged. • <*RemainingWork*> (Long): Sum of the values for <*FieldsFlagged*> + `FieldsUnconfirmed` + <*FieldsInvalid*> + <*CharactersInvalid*> + [number of fields failing Document-level validation]. A value of zero means the

IA Value	Level	Type	Description
			document should be complete and ready to export. Processes can use this to determine if the document has any remaining work rather than manually adding up the individual values in a CaptureFlow Designer expression. If this value is nonzero, the document contains work that can be corrected by a step configured for the Document work level. It might also contain work applicable to lower work levels; check the other status values to find out. **Note:** Fields may be counted multiple times if multiple states apply. For example, if a field is both in error and has a flag, <*FieldsInvalid*> is 1, <*FieldsFlagged*> is 1, and <*RemainingWork*> is 2, although only one field needs work. Routing

IA Value	Level	Type	Description
			decisions based on information in the document status should take this into account.
<ExternalValues>	0	String	Contains a list of IA Values to be used as external values from the Extraction module in the recognition VBA scripting. To assign the list of IA Values, create the *<ExternalValues>* custom value for Extraction (level 0) in the CaptureFlow. Then specify the comma-separated list of required IA values enclosed in double quotes. For example, to obtain the FolderID and MachineID values, in the CaptureFlow for Extraction set them as follows: `Extraction:0.ExtervalValues = "FolderID, Mac`
<Image>	0	File, Input	The input image, generally a `TIF` file in the color and compression format output by the preceding CaptureFlow step.

IA Value	Level	Type	Description
<InputPageDataXml>	0	String, Input, NoTrigger	*XML* data generated by the preceding step, such as Classification or Identification, and passed in to the Extraction step. Pre-indexed fields already recognized in Classification and validated in Identification are stored in this XML object.
<OcrDataCache>	0	File, Input, Output	As an input, the optional full text *OCR* data cache, typically created by Classification. After extraction, contains the Extraction-created OCR data on output.
<OcrLocale>	1	String, Input	Contains the locale, for example *<de-DE>* that will be used by Information Extraction for extracting text from the images. If empty, the locale is defined by the first locale in the Information Extraction profile for the selected document type.
<OcrEngine>	1	String	Select the OCR engine for the extraction request. The accepted values are ***GOOGLE*** and ***CRE***. You can use these OCR engines only when you enable classification with information extraction-based projects. The default value is ***CRE***.

IA Value	Level	Type	Description
<*OutputPageDataXml*>	0	String, Output, NoTrigger	XML data produced by the Extraction module. This XML data serves as input to the Completion module. In a typical CaptureFlow, <*OutputPageDataXml*> is bound to <*PageDataXml*> in Completion.

IA Value	Level	Type	Description
<*PageTemplateId*>	0	String, Input, NoTrigger	The page template ID that will be placed in the <*OutputPageDataXml*> value and used for data extraction. Values: • Greater than "0": The specified value is a valid page template ID (plain non-xml text) that must be used for data extraction. This value overrides the template ID passed in the <*InputPageDataXml*> value or in the document type. • "0" (zero) or less than "–1": The template ID is determined from the <*InputPageDataXml*> value or the document type. In this case, if detected that a count of templates of the document type is less than a count of pages in the document, then: – for <*PageTemplateId*> equal to "0", the remaining pages of the document are skipped in Extraction; – for <*PageTemplateId*> less than "–1", the

IA Value	Level	Type	Description
			remaining pages of the document are assigned to the last template of the document type defined in the <*UimDocumentType*> IA Value. • "–1": The page must be skipped from data extraction. If <*PageTemplateId*> is not set (empty), it is considered to be "0" (zero). Consider showing a valid template ID if Extraction gets an empty input XML (<*InputPageDataXml*>), for instance, when Classification and Identification are not used in a CaptureFlow. Extraction will create <*OutputPageDataXml*> automatically, based on the specified template ID (<*PageTemplateId*>) and without <*InputPageDataXml*>.

IA Value	Level	Type	Description
<*PagesToProcess*>	1	String, Input	This value holds a comma-delimited string with page numbers to be extracted. For example: • "1": Extraction is performed on only page 1. • "3–5": Extraction is performed on pages 3, 4, and 5. • "L2": Extraction is performed on the last two pages. • "1,3–5,L2": Extraction is performed on pages 1, 3, 4, 5, and the last 2 pages. If this value is empty, then extraction is performed on all pages.
<*ReportTag1*>	Task	String, Input	Used for statistical data.
<*ReportTag2*>	Task	String, Input	Used for statistical data.
<*ReportTag3*>	Task	String, Input	Used for statistical data.
<*StepCustomValue*>	T	String, Output	Used as scripting parameters. If set to a non-empty value, overrides those values defined in the step setup.

IA Value	Level	Type	Description
<UimData>	1	String, Output	Information extracted from the document. Within the batch, every extracted document (level 1 node) holds the document type object with extracted data in this value. The document type object produced by the Extraction module serves as input to the Completion module that uses this object to identify the document type and index fields for the document.

IA Value	Level	Type	Description
<*UimDataImportMode*>	1	Long, Input, Output	The mode that defines or disallows additional data import to <*UimData*>. Input data is bound to the step regardless its index level. Allowed values are: • 0 - Do not bind additional data to <*UimData*>. • 1 - One-timely bind dynamic field data input to <*UimData*>. The data should be in values <*InUimData_field*> or <*InUimData_field__Index*>. • 2 – Always bind index values higher than the task level from the flattened output IA values back to <*UimData*>. The data should be in values named exactly as their target fields: <*field*>. When the field is an array (table column), all array items are expected, separated with a new line. Additionally, the "per row" array values are supported: <*field*>_<*row*>. In case IA values contain both "per array" and "per row" flattened data, the last one takes precedence.

IA Value	Level	Type	Description
			• 3 – The combination of modes 1 (“one-timely bind”) and 2 (“always bind”). After a value has been copied into the <*UimData*> object, <*UimDataImportMode*> is reset to “2”. In case of conflict between prefixed “one-time” and undecorated “always” values, the last one takes preference. To learn more about data flattening and passing data between IA values and UIM data, see *OpenText Intelligent Capture - Designer Guide (ECPCORE-CPD)*.

IA Value	Level	Type	Description
<*UimDocumentType*>	1	String, Input	The name of the document type. You can use this value to assign a particular document type to the Extraction step directly. Otherwise, Extraction will use the page template (identified and passed in by a preceding step, such as Classification) to determine the document type/index family for data extraction. When working with an IEE project, you must set the level 1 value to specify the document type to use. IEE does not work at a page level, so you do not have to specify level 0 values.

Chapter 3

NuanceOCR

This section includes the Intelligent Capture NuanceOCR module: **ecpcore-cnu**.

3.1 Introduction

NuanceOCR performs optical character recognition of scanned or imported images. It exports the image and index data to more than 25 different word processing and text formats.

Specifically, NuanceOCR can perform the following tasks and functions:

- **Obtains high-quality *OCR* using multiple recognition engines**: NuanceOCR uses several different recognition engines to obtain the highest quality recognition possible.
- **Handles multiple recognition languages simultaneously**: Lets you select the recognition languages from an extensive list. Multiple languages can be selected to perform accurate recognition of entire pages in a single pass.
- **Checks and corrects the spelling of recognized pages**: Enables spell checking in a single language, increasing the overall accuracy of the recognized text.
- **Adds characters to the current set of recognition languages**: When setting up NuanceOCR, you can define a list of characters that are not in the character sets of the selected language. For example, adding é to the *US* English character set allows words such as résumé to be properly recognized and preserved in the output document.
- **Offers various output code pages for specific export needs**: Users can select the code page to use when exporting recognized documents. Typically, there are different code pages for different languages. For example, ASCII code page 850 supports English and Western European languages. The Windows Western *ANSI* code page also supports these languages.
- **Uses filter combinations to constrain recognition**: Filter recognition limits recognized text to a restricted range of characters. For example, this function can improve recognition results when working with pages that only contain numbers and related symbols. Results are obtained by combining filters that accept digits only, lowercase letters only, punctuation only, uppercase letters only, and miscellaneous characters.
- **Supports many output formats**: NuanceOCR supports 39 output formats, including *HTML* and *PDF*.
- **Exports multiple formats in a single pass**: NuanceOCR can be configured to export multiple document formats in a single pass. By default, it is possible to configure two separate formats. To add more formats, edit the *MDF* to include a set of IA values for each additional format.

- **Outputs documents to the Intelligent Capture Server or directly to disk**: NuanceOCR lets you route recognized document files back to the Intelligent Capture Server for further processing, export them directly to the file system, or both. Exporting directly to a file saves the work of including and configuring an export module during processing. Exporting to the Intelligent Capture Server enables additional processing steps with other modules before exporting.

3.2 Setting Up NuanceOCR

Most configuration information for Intelligent Capture client modules is defined during *module setup*. You can run a module in setup mode from Intelligent Capture Designer, from Intelligent Capture Administrator, or from a command prompt with appropriate arguments. For more information, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

When launched in setup mode, a client module displays the **Setup** window in which you can specify the module's configuration settings.

3.2.1 Understanding Recognition Engine Overrides

Engine overrides control the behavior of the *OCR* engines that are loaded into memory when running the module. The **Time Out** option can be set to limit the length of time the engine uses to recognize a page. Or, an **Unload Engine** interval can be used to improve performance.

- **Time Out**: Some pages can cause long recognition delays, such as a graphical page that NuanceOCR attempts to recognize, but cannot do so in a reasonable time. Time out specifies the number of minutes or seconds the OCR engine processes pages before abandoning the page and moving to the next page.
- **Unload Engine**: Without specifying otherwise, NuanceOCR keeps its recognition engines loaded in memory as long as the module is running. OCR technology can significantly consume system resources, so NuanceOCR can be set up to unload the engines occasionally. However, unless there is a significant performance reduction, unloading and reloading the engines too frequently can take additional time. To avoid memory issues, the optimal setting (which is also the default) is 0 **Pages**, and settings above 3,500 **Pages** are not recommended.

  

  Note: Restart NuanceOCR for the changes made to the **Unload engine** option to take effect. NuanceOCR reads these options during module startup.

- **Restart module if an error occurs**: Whether to restart the NuanceOCR module if an error occurs.
- **Preprocess images**: Specify one of the following:

  – Cleared

    Does not perform image format conversion and uses the NuanceOCR engine directly, which might improve recognition accuracy and module throughput.

– Selected

Performs image format conversion, which is CPU-intensive and reduces module throughput.

Notes

– The NuanceOCR engine supports a subset of the Intelligent Capture supported image formats.

3.2.1.1 Configuring Recognition Engine Overrides

To configure recognition engine overrides:

1. Run NuanceOCR for setup.
2. Select **Engine** from the navigation panel.
3. Under **Time Out** in the **Engine Settings** pane, select either **Minutes** or **Seconds**. Type the number of minutes or seconds to wait before abandoning the current page and moving to the next.
4. Under **Unload Engine**, select either **Tasks** or **Pages**. Type the number of tasks or pages NuanceOCR can process before unloading and then reloading the engine. To avoid memory issues, the optimal setting is 0 **Pages**, and settings above 3,500 **Pages** are not recommended.

   Note: Restart NuanceOCR for the changes made to the **Unload engine** option to take effect. NuanceOCR reads these options during module startup

5. Click **Apply** to save these changes and make additional changes in other panes, or click **OK** to save the changes and exit the setup window.

3.2.2 Understanding Document Recognition Options

Document recognition settings are the basis for character recognition. Specify one or more recognition languages, along with spelling languages to apply spelling correction. Use the **Document Recognition Settings** pane to specify how the module performs recognition on each page.

Note: The NuanceOCR optical character recognition engines can impose image size limitations. The size limitation varies depending on such factors as image width and height, color resolution, compression, and available memory. Under most circumstances, this limitation is not an issue. However, if the image size exceeds the limits, NuanceOCR returns an unsupported image size error. In this case, adjust the project parameters to eliminate the error.

The available settings in the **Document Recognition Settings** pane are:

- **Rotation**

- **Auto-rotate image before recognition**: Select to have NuanceOCR automatically correct the orientation of images before performing recognition. If your images are already in the correct orientation, or if using another module to correct image orientation, clear this check box to improve overall module throughput.

  Note: Hebrew does not support auto-rotation.

- **Save rotated image on server**: Select to save the image that was used for recognition to the Intelligent Capture Server, including any automatic rotation that the module performed. This option is available only if **Auto-rotate image before recognition** is enabled.
- **Rotated image format**: Specify the file's rotated image format in which to save to the Intelligent Capture Server.

  Note: This option is enabled only if **Save rotated image on server** is checked.

- **Character Recognition**

  - **Recognition languages**: Select the languages that are present on the pages being processing. If needed, select multiple recognition languages. Select once to select a language, select again to clear it. By default, a single recognition language is selected to match the regional settings of the operating system. For a list of languages supported by NuanceOCR, see "Supported Languages" on page 71.

    Notes

    - Select **Install files for East Asian languages** in the Windows **Regional and Language Options** so that NuanceOCR can properly recognize and output documents containing Chinese, Japanese, or other East Asian characters.
    - Select only one Asian language for recognition at a time, and do not select any Western languages at the same time as an Asian language. The Asian *OCR* Engine can recognize short English phrases embedded in Asian text without English being selected. If embedded phrases are in other Latin-alphabet languages, you do not need to select these languages; however, accented characters are sometimes not recognized correctly.

  - **Spelling language**: Select a single spelling language to apply to tasks.
  - **Enable automatic spelling correction**: Select to perform an automatic spell check of the recognized text in the selected spelling language. Clear this check box to skip spell-checking. Checking spelling takes additional time for each page processed, but results in higher accuracy.
  - **Additional characters**: Type additional characters that are not part of the character set of the selected recognition languages. These additional characters are added to the recognized character set for the currently selected

languages. For example, with English it is possible to recognize the é that is often used in words like résumé. Type this character in the field.

- Enter characters that are not available on the keyboard by using the Windows Character Map program. Or, type codes on the numeric keypad while holding down **ALT**. For example, **ALT+0233** is the numeric key code for é.
- Specify multiple characters, symbols, or punctuation in a single group with no delimiters.

– **Substitute this character for unrecognized characters**: NuanceOCR assigns a confidence code to each character and word that it recognizes. If the confidence of an individual character is below a preset threshold, the character is marked as "unrecognized". In this field, specify the character to insert for each unrecognized character it processes. Later, after the document capture operation is finished, open the resulting document and search for instances of this character.

- **Code Page Definition**

  – **Code page**: The code page defines the character set for the document. Select the code page that most closely matches the languages to recognize. Selecting **Automatic (AUTO)** for the code page causes the module to select a code page appropriate for the languages that are being recognized.

  – **Type**: The definition type that is applied.

  – **Description**: The description of the **Code page** type.

  – **Substitute this character for characters not found in the code page**: Specify a unique character to replace any unrecognized characters.

3.2.2.1 Configuring Document Recognition Settings

Before setting up zones and defining details of character recognition, configure the document recognition settings.

To configure document recognition settings:

1. Run NuanceOCR for setup.
2. Click **Document recognition** from the navigation panel and the **Document Recognition Settings** pane displays.
3. Select **Auto rotate image before recognition** if image rotation is necessary. Select **Save rotated image on server** to save a copy of the rotated image.
4. Select the **Character Recognition** options:

   - **Recognition languages**: Select the languages that are present on the pages being processed. For a list of languages supported by NuanceOCR, see "Supported Languages" on page 71.

- **Spelling language**: Select a single spelling language to apply to tasks. If the **Enable automatic spelling correction** check box is selected, the module spell-checks the recognized text in the selected language. If recognizing a language that is not in the **Spelling Language** list, clear this check box.
- **Additional characters**: Type additional characters that are not part of the character set of the recognition languages.
  - Enter characters that are not available on the keyboard by using the Windows Character Map program. Or, type codes on the numeric keypad while holding down **ALT**. For example, **ALT+0233** is the numeric key code for é.
  - Specify multiple characters, symbols, or punctuation in a single group with no delimiters.
- **Substitute this character for unrecognized characters**: Specify the character to insert for each unrecognized character it processes.

5. Select the **Code page definition** options:
   - Select the **Code page** that most closely matches the languages to recognize. Selecting **Automatic (AUTO)** for the code page causes the module to select a code page appropriate for the languages that are being recognized.
6. Either click **Apply** to save these changes and make additional changes in other panes, or click **OK** to save the changes and exit the setup window.

3.2.3 Creating Recognition Zones

You can perform full page recognition on documents, or define a number of zones over the documents and define custom recognition settings for each defined zone. For more information on zone definitions, see "Understanding Recognition Zone Settings" on page 45.

By default, up to 50 zones can be defined automatically in NuanceOCR. Additional zones are created in two ways:

- By adding more zones in the `SSOCR.MDF` file using the format, *<Level0_ZoneValue<51>_RecText>* and *<Level0_ZoneValue<51>_RecText>*.
- By using the rubber band operation (described in the following procedure), additional zones are manually created. They are dynamically added as IA values for consideration during production. These dynamic zones are not displayed as nodal items in Intelligent Capture Administrator however.

To create recognition zones:

1. Run NuanceOCR for setup.
2. Click **Zone Definition** from the navigation panel and the **Zone Definition Settings** pane displays.

3. Click **Set Up Zones**. The **Select Sample Image** window displays for opening a sample image. Locate and open the sample image. Select an image that is not skewed and that is a good representation of the images processed that this module.
4. Click **Open** and the **Zone Setup** window displays the sample image in the **File view** pane.
5. In the **Zone Actions** panel, set the following options:
   - **Find Zones** automatically locates zones on the image. Alternatively, left-click and drag the mouse ("rubberband") over the image to draw a zone manually.

     Note: When using **Find Zones**, NuanceOCR only defines the number of zones (50 by default) defined in `SSOCR.MDF`. To define additional zones, use the rubber band operation. Or, to define more than 50 zones automatically, add additional zone definitions in the *MDF* file as described in the introduction of this topic.
   - **Full-page zone** expands the selected zone to cover the entire page during recognition. Only one full-page zone can be defined per image.

     Note: When defining a full-page zone and additional zones, the recognition phase of the module occurs twice. It occurs a first time to retrieve the full-page information and a second time to retrieve the other zone data. If a full-page zone is defined, it is used to produce any output files.
   - **Delete Zone** to remove the selected zone.
   - **Up** or **Down** changes the order position of the selected zone.
6. After zones are defined, select them and set custom settings for each zone. For more information, see "Defining Custom Zone Settings" on page 49.

3.2.4 Understanding Recognition Zone Settings

Recognition zone settings control the behavior of NuanceOCR during character recognition. Default zone definitions are defined in the **Zone Definition Settings** pane of the **Intelligent Capture NuanceOCR Setup** window. Use default settings for character recognition, or customize the settings for each zone from the **Zone Setup** window. *OMR* (Optical Mark Recognition) settings are defined in the **Optical Mark Recognition** section of the **Zone Definition Settings** pane.

3.2.4.1 Defining Default Zone Settings

The **Zone Definition** pane enables defining of default settings for all zones. For more information on creating zones, see "Creating Recognition Zones" on page 44. The default settings are applied to all zones for the batch unless custom zones and settings are defined for individual zones. For more information, see "Defining Custom Zone Settings" on page 49.

To define default settings for all zones:

1. Run NuanceOCR for setup.
2. Click **Zone Definition** from the navigation panel and the **Zone Definition Settings** pane displays.
3. In the **Zone Defaults** field, select **Enable OCR-assisted indexing**, and then select **ASCII PDA** or **XML PDA** to write the output to a file and save this information to the server as <*Level0_OutputPDA*>.

   Note: Some languages contain characters that have codes outside the *ANSI* character set (codes greater than 255). Select **Enable OCR-assisted indexing** and select the **XML PDA** format when processing pages containing these languages. For a list of languages supported by NuanceOCR, see "Supported Languages" on page 71.

4. From the **Available filters** list, select the filters to apply to the zone. Selecting no filters enables all possible characters and words to be recognized. Selecting one or more filters limits recognition to the characters and symbols defined by the filters. The available filters include:

   - **DIGIT**: Recognizes any combination of digits 0 through 9 on the page.
   - **LOWERCASE**: Recognizes lower-case letters, including accented ones.
   - **MISCELLANEOUS**: Recognizes miscellaneous non-alphabetic, non-numeric, and non-punctuation characters such as "+".
   - **PUNCTUATION**: Recognizes the punctuation symbols such as "?", "!", and";"
   - **UPPERCASE**: Recognizes uppercase letters, including accented ones.

5. Click **Add** to add the selected filters to the **Selected filters** list. From the **Selected filters** list, select the filters to delete from the list, and then click **Remove**.
6. From the **Recognition engine** list, select the recognition engine to use as the default for the defined zones. For more information on available engines, see "Recognition Engines and Filling Methods" on page 73.

   Note: Depending on the feature codes associated with your license, some recognition engines are not available.

   The available engines include:

- **Automatic (AUTO)**: The module analyzes each page to determine the best engine or combination of engines to apply. This engine is recommended for most normal use of the module.
- **Machine Print (OMNIFONT_MTX)**: Optimized for omnifont character recognition in English.

  Note: The recognition engine Machine Print (OMNIFONT_MTX) supports the characters from a limited set of languages. For more information on supported languages, see "Machine Print (OMNIFONT_MTX) Supported Languages" on page 72
- **Multi-Language Machine Print (OMNIFONT_MOR)**: Optimized for multilingual omnifont character recognition.
- **9-pin Dot Matrix (DOT)**: This engine is designed for recognizing only draft-quality 9-pin dot-matrix texts.
- **Barcode module (BAR)**: This engine is designed for recognizing 1D and 2D barcodes. For a list of supported barcode types, see "Supported Barcode Types" on page 74.
- **OMR optical mark (OMR)**: This recognition engine is used for recognizing optical marks. Typical application areas are where the documents are form-like and are filled out by hand such as questionnaires, educational tests, and reporting or order sheets.

  Note: The *OMR* engine does not perform character recognition automatically. Define zones manually.
- **Matrix Matching (MAT)**: This recognition engine is designed to recognize certain groups of fixed-font characters specially designed for *OCR* or imaging applications where no two characters have similar shapes. Each character group has its own filling method.
- **2-Way Voting (PLUS2W)**: Recognizes machine printed text from printed publications, laser or ink-jet printers, and electric typewriters. This recognition engine is a combination of *MTX* and *MOR* engines. It includes voting to achieve up to 40% fewer errors. Voting consumes up to 50% more recognition time.
- **3-Way voting (PLUS3W)**: Recognizes machine printed text from printed publications, laser or ink-jet printers, and electric typewriters. This recognition engine is a combination of *MOR*, *MTX*, and *FRX* engines.
- **Asian recognition module (ASIAN)**: Recognizes Asian languages.

7. From the **Trade off** list, select the speed versus accuracy trade off default:

   - **ACCURATE**: Spend more time recognizing the page to produce a more accurate result.
   - **BALANCED**: Balance recognition speed with accuracy.
   - **FAST**: Spend less time recognizing the page at the expense of accuracy.

8. From the **Filling method** list, select the content that is filling the zone. For more information on filling methods available based on the recognition engine selected, see "Recognition Engines and Filling Methods" on page 73.

   - **Machine Print (OMNIFONT)**: Machine-printed text with any fonts that are not too stylized.
   - **9-Pin Draft Dot-Matrix (DRAFTDOT9)**: Draft-quality output from a 9-pin dot-matrix printer.
   - 1D barcode: Linear barcodes.
   - **Optical Mark**: This method handles any mark in a zone. It is used mainly for defining marks (filling in the answer bubble on some tests), check marks on questionnaires, and signatures.
   - **24-Pin Draft Dot-Matrix (DRAFTDOT24)**: Draft-quality output from a 24-pin dot-matrix printer.
   - **OCR-A**: The character set contained in ANSI Standard X3.17-1981. A stylized font for traditional OCR printing.
   - **OCR-B**: The character set contained in ANSI Standard X3.49-1975. A stylized font for traditional OCR printing.
   - **Magnetic Ink (MICR)**: Characters that are readable by an MICR machine.
   - **2D barcode**: Two-dimensional matrix barcodes.
   - **Dot-Digit**: Ten digits and the period. Commas are read, but converted to periods.
   - **Dash-Digit**: Ten digits only and the period. Commas are also read, but converted to periods.
   - **Asian languages (ASIAN)**: Asian character sets.
   - **No OCR**: Do not OCR. If you apply this method to a zone, the OCR engine will not OCR the zone. This method can be useful for graphics where you do not want to recognize the text or other characters in the graphic.

9. Select options in the **Frames** list in the **Optical Mark Recognition** field. Options indicate whether there is a box or a circle around the location for the optical mark:

   - **Automatic** : The module automatically determines whether there is a frame around the location.
   - **No**: This option indicates that there is no box or circle around the location for the check.
   - **Yes**: This option indicates that there is a box or circle around the location for the check. If a frame exists, this option results in better accuracy than selecting Automatic.

10. From the **Sensitivity** list in the **Optical Mark Recognition** field, choose one of the following options to indicate the level of tolerance for the recognition engine when detecting marks.

- **Normal**: Highest mark sensitivity. **Normal** is the default value.
- **Low**: Less mark sensitivity.
- **Lower**: Lower mark sensitivity.
- **Lowest**: Lowest mark sensitivity.

The higher the mark sensitivity setting, the more sensitive the recognition. For example, even a small mark (just a few contiguous black pixels) causes the zone to be classified as "checked by respondent". The default sensitivity is suitable for good quality printed questionnaires. Lower sensitivity allows speckled or poorly printed documents to be processed successfully. Marks are returned in zones as 0 or 1. If there is no mark within the zone, the text for the zone is 0. If there is a mark within the zone, the text for the zone is 1.

11. Click **OK** to close the window and save the settings to the server.

3.2.4.2 Defining Custom Zone Settings

When creating zones in the **Zone Setup** window, you can customize the zone settings for each defined zone. The settings defined here override the default settings from the **Zone Definition Settings** panel. Zones can also intersect.

To define custom zones from the Zone Setup window:

1. Run NuanceOCR for setup.
2. Click **Zone Definition** from the navigation panel and the **Zone Definition Settings** pane displays.
3. Click **Set Up Zones**. If an image is not currently selected, the **Select Sample Image** window displays. Locate and open the sample image. Select an image that is not skewed and that is a good representation of the images processed by this module.
4. Click **Open** and the **Zone Setup** window displays the sample image in the **File view** pane. For more information on creating zones, see "Creating Recognition Zones" on page 44.
5. Select a zone or zones on which to specify custom settings. On the **Recognition Settings** panel, set the following options:
   - For **Zone type**, specify **Flowed Text** or **Graphic** to ignore the selected zone. When selecting **Graphic**, the zone on the sample image displays as red to distinguish this selection. The **Vertical text** zone type is available for Asian recognition languages only.
   - **Recognition engine** specifies the type of printout that is being analyzed. This works with the **Filling method** setting.
   - **Filling method** specifies the type of content that is filling the zone. This works with the **Recognition engine** setting, and is automatically reset to reflect the current **Recognition engine** selection. For more information, see "Recognition Engines and Filling Methods" on page 73.

- **Disable spelling language** disables the use of the language dictionary. Selecting this check box overrides the setting of the **Zone Definition Settings** panel.

6. On the **Filters Settings** panel, set the following options:

   - **Use default filters** uses the filters defined on the **Zone Definition Setting** pane.
   - **Use these filters** enables override of defaults by selecting the filters for the selected zone. These filters limit the type of recognized characters. Selecting no filters enables all possible characters and words to be recognized. Selecting one or more filters limits recognition to the characters and symbols defined by the filters.

7. The **IA Values** panel allows specification to return recognized test information as IA values.

8. To change the sample image, type the path and file name of the sample image to open an image in the **Sample Image** field. Or, click **Browse** to navigate to the file. The currently defined zones remain for the sample image. If necessary, click **Set Up Zones** to create a set of zones.

9. After configuration of all **Zone Setup** options, click **OK** to close the window and return to the **Zone Definition** pane.

10. Select the **Add zones before rotate** check box to set whether recognition zones are applied to images before or after the image is rotated.

    Notes

    - For this option to be enabled, at least one zone must be configured.

11. Click **Apply** to save these changes and make additional changes in other panes, or click **OK** to save the changes and exit the setup window.

3.2.5 Selecting Output Formats

The **Output Formats Settings** pane of the **NuanceOCR Setup** window enables selection of the text or word-processing output format of the module. By default, output is limited to two formats which are output to the Intelligent Capture Server, a disk file, or both. Each output format can have its own, independent settings based on the controls in this pane. To specify additional formats beyond the two allowed, edit the *MDF*, and specify a set of values for each additional output format. If no output formats are defined, a *PDF* output file can be created for indexing. For a list of available formats, see "Output Formats" on page 68.

3.2.5.1 Defining Output Formats

Define output formats and format settings for the recognized text.

To define output formats:

1. Run NuanceOCR for setup.
2. Click **Output formats** and the **Output Formats Settings** pane displays.
3. Click **New Format** to create the first format. To specify a second format, click the button again.
4. In the **Format** drop-down box, select an output format.
5. In the **Level** drop-down box, select one of the following options:
   - **Auto**: Uses the default output mode of the selected converter.
   - **NOFORMAT**: Existing formatting information is ignored and is output as a single column, left-aligned paragraphs, no font attributes, etc.
   - **RFP**: Retains fonts and paragraphs.
   - **TRUEPAGE**: Highest layout accuracy maintained with text boxes or frames, but not for serious editing tasks.
   - **FLOWINGPAGE**: Preserves the original layout of the pages, including retaining columns. The boxes and frames are only used when necessary.
   - **SPREADSHEET**: Exports the results in tabular form, suitable for use in spreadsheet applications. Each page is placed in a separate worksheet.

   Note: The text converters specified by the selected **Format** also support various **Levels**. The "Supported and Default Formatting Levels" on page 70 section explains how to determine which level to use.
6. Click **Test** to test the selections. For more information, see "Testing Recognition Settings" on page 53.
7. Select **Save to server**, **Save to file system**, or both.
8. When **Save to file system** is selected, specify these additional parameters.
   - **Path**: The location where the file is written. Type the location or IA value to use for the path, click **Browse** to locate the folder, or click **Insert Value** to specify an IA Value.
   - **File**: The file name to use. The field to the right of the text box indicates the format and extension of the file. Click **Insert Value** to use an IA value to create the file name.
   - **If the file exists, then**: The action to take for an existing file. The **Prompt for an action** option cannot be used when the module run as a service. This option requires user input.

9. In **PDF output settings**, specify these options:

   - **Color quality**: Set the *PDF* color quality.

     Notes

     – This option is only available for the following output formats:

     ○ **Adobe PDF with image on text**

     ○ **PDF/A with image on text**

   - **PDF compatibility**: Set the version with which the PDF or PDF/A is compatible. This option is only available for the following output formats.

     – For PDF version compatibility:

     ○ **Adobe PDF**

     ○ **Adobe PDF edited**

     ○ **Adobe PDF with image on text**

     ○ **Adobe PDF with image substitutes**

     – For PDF/A version compatibility:

     ○ **Adobe PDF [PDF/A]**

     ○ **Adobe PDF edited [PDF/A]**

     ○ **Adobe PDF with image on text [PDF/A]**

     ○ **Adobe PDF with image substitutes [PDF/A]**

   - **Use JBIG2 compression**: Specify whether to use the JBIG2 compression method for PDF files.

     Notes

     – This option is available for the following output formats only:

     ○ **Adobe PDF with image on text**

     ○ **PDF/A with image on text**

   - **Web optimized PDF**: Specify whether to optimize PDF files for the Web.

     Notes

     – This option is available for all of the Adobe PDF output formats on the **Output Formats** pane.

10. Click **Apply** to save these changes and make additional changes in other panes, or click **OK** to save the changes and exit the setup window.

3.2.6 Testing Recognition Settings

To test recognition settings before running the module in production mode, use the **Test** button in the **Output Formats** pane of the **NuanceOCR Setup** window. Testing the setup enables experimentation with different settings until finding the best options. Modifying the settings in the **Test** window actually changes the module setup.

To test recognition settings:

1. In the **NuanceOCR Setup** window, select **Output Format**.
2. Select an output format, then click **Test**. The **Test** window displays.
3. Under **Input Files**, click **New Input** and specify up to five test files.
4. Under **Output File**, specify a path and file name for an output of the recognized text during the test.
5. The **Recognition Settings** are based on the current selections from the other setup panes. These settings can be modified from this location while fine-tuning the testing. If necessary, adjust these options.
   - ***Additional characters***: Type additional characters that are not part of the character set of the recognition languages.
     - To enter characters that are not available on the keyboard, either use the Windows Character Map program. Or, type codes on the numeric keypad while holding down **ALT**. For example, **ALT+0233** is the numeric key code for é.
     - Specify multiple characters, symbols, or punctuation in a single group with no delimiters.
   - **Recognition engine**: Specifies the type of printout that is being analyzed. This works with the **Filling method** setting.
   - **Filling method**: Specifies the type of content that is filling the zone. This works with the **Recognition engine** setting, and is automatically reset to reflect the current **Recognition engine** selection. For more information, see "Recognition Engines and Filling Methods" on page 73.
   - Select a **Format level**:
     - **AUTO**: Uses the default output mode of the selected converter.
     - **NOFORMAT**: Existing formatting information is ignored and is output as a single column, left-aligned paragraphs, no font attributes, etc.
     - **RFP**: Retains fonts and paragraphs.
     - **TRUEPAGE**: Highest layout accuracy maintained with text boxes or frames, but not for serious editing tasks.
     - **FLOWINGPAGE**: Preserves the original layout of the pages, including retaining columns. The boxes and frames are only used when necessary.

– **SPREADSHEET**: Exports the results in tabular form, suitable for use in spreadsheet applications. Each page is placed in a separate worksheet.

Note: The text converters specified by the selected **Format** also support various **Levels**. The "Supported and Default Formatting Levels" on page 70 section explains how to determine which level to use.

- Select the **Code page** that most closely matches the languages to recognize. Selecting **Automatic (AUTO)** for the code page causes the module to select a code page appropriate for the languages that are being recognized.
- From the **Trade off** list, select the speed versus accuracy trade off option.

6. Under **Actions**, click **Run** to test the current selections. An **Output File** must be specified to enable this option. The **Status** pane shows processing statistics after testing has begun. The **Image View** pane shows the image that is selected in the **Input Files** list.
7. Click **View** to evaluate the output text file.
8. Adjust settings in this window and run the test again. If necessary, changing the settings in the **Test** window can change the module setup.
9. Click **Apply** to preserve the changes and remain in the test window. Click **OK** to save the current settings and close the **Test** window, or click **Cancel** to discard any changes made in the test window.

3.3 Running NuanceOCR in Production

After the module has been set up, tested, and placed in a production environment, operators and administrators will run the module in production.

For information related to common production tasks for unattended modules, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

Note: The *Running Modules in Production Mode* section contains production topics that are common to many unattended modules and may not apply to this module.

3.4 Reference—NuanceOCR

The topics within this section contain reference information useful while using the application in setup or production.

3.4.1 Programming Reference

Scripting interfaces are provided as .NET libraries. Scripts are written using standard .NET programming languages such as VB. NET (Visual Basic) and C#. Find scripting samples in the Intelligent Capture program files folder, located by default at `<<drive>>:\Program Files (x86)\InputAccel\Client\src\Sample Capture System\ScriptSource\Client-side Scripts\`.

For more information about scripting, see *OpenText Intelligent Capture - Scripting Guide (ECPCORE-PSC)*.

3.4.2 IA Values

The topics in this section describe IA values that can be used with this application.

3.4.2.1 Input IA Values

Input IA values include input file variables, such as <*Level0_InputFile1*>, which serves as a pointer to stage files. They also include input processing variables, which store setup data that a module uses to process tasks, including default and user-defined module settings.

Input IA Values

Input file variables

<*Level0_InputImage***>**
The task's input image on the Intelligent Capture Server. <*InputImage*> may be any type of binary, grayscale, or color image file processed by upstream modules. If this value is used in the *IPP*, it is a trigger. When all trigger values are non-zero, the module will process the current task.

- **Attributes**: File, Input, Trigger
- **Level**: 0

Input values

<*Ready***>**
A secondary trigger value that can be used when no file values are specified or when another trigger is needed to adequately control the beginning of processing. If this value is used in the IPP, it is a trigger. When all trigger values are non-zero, the module will process the current task.

- **Attributes**: File, Input, Trigger
- **Level**: T

<*Level0_Processed***>**
Valid values are:

- 1: The node has already been processed, and may be skipped if the module is configured to skip processed nodes on retry after an error.
- 0: The node has not yet been processed.

IA value attributes are:

- **Attributes**: Long, Input, NoTrigger
- **Level**: 0

The following variables can be changed as dynamic IA values in an IPP, Intelligent Capture Administrator, or scripting.

Note: These setup values are available for editing in Intelligent Capture Administrator only after the NuanceOCR step has been configured at least once.

<*_ForceTestRecognition***>**
For tuning auto-rotation.

0 (default): The module attempts to detect the orientation of the image and applies test recognition on a small part of the image only if the orientation was not detected.

1: Auto-rotation always performs test recognition.

3.4.2.2 Output IA Values

There are several types of output IA values returned by the NuanceOCR module. Output IA values include the output files, the values describing output files, status and error values, statistical values, and zone output values. Output files serve as pointers to stage files.

Output IA Values

Output files

<*OutputFile1_OutputFile***>**
The first of as many as two output files from the task. The file that corresponds with <*OutputFile1_OutputFile*> can be any file type the module is capable of outputting. The output file type is configured in the **Output Formats** pane of the **NuanceOCR Setup** window.

- **Attributes**: File, Output
- **Level**: T

<*OutputFile2_OutputFile***>**
The second of as many as two output files from the task. The file that corresponds with <*OutputFile2_OutputFile*> can be a file of any type the module

is capable of outputting. The output file type is configured in the **Output Formats** pane of the **NuanceOCR Setup** window.

- **Attributes**: File, Output
- **Level**: T

Note: Add additional output files (enabling more than two simultaneous output files per task) by adding the entire set of *<OutputFile<n>_>* values, where *<n>* represents the sequential number of each section. For example, to add a third output file, add the following IA values to the *MDF*: *<OutputFile3_OutputFile>*, *<OutputFile3_FullPath>*, *<OutputFile3_Ext>*, *<OutputFile3_Created>*, *<OutputFile3_ErrorNumber>*, and *<OutputFile3_ErrorText>*. The number of output formats enabled in the MDF is automatically reflected in the **Maximum Formats Allowed** label on the **Output Formats** pane of the **NuanceOCR Setup** window.

<*Level0_OutputPDA***>**
Path to the tasks Process Document Architecture file (*PDA*) on the Intelligent Capture Server. The corresponding PDA contains the recognized text and selected pieces of the original image processed by NuanceOCR.

- **Level**: File, Output
- **Level**: 0

<*Level0_RotatedImage***>**
The automatically-rotated image that the module used for recognition. This file exists only if **Save rotated Image on server** was enabled during setup, and only if *<Level0_Rotation>* is not 1 during production; that is, only if the image was automatically rotated by the module.

- **Attributes**: File, Output
- **Level**: 0

`Output files`

<*Level0_Rotation***>**
The orientation of the original image. If **Auto-Rotate image before recognition** was enabled during setup, then during production, this value indicates the orientation of the *<Level0_InputImage>* value. Valid values are:

- 1 = 0 degrees
- 2 = 90 degrees
- 3 = 180 degrees
- 4 = 270 degrees

IA value attributes are:

- **Attributes**: Long, Output
- **Level**: T

<*Level0_RotatedImgFormat***>**

The format of <*Level0_RotatedImage*> that the module saved to the server.

When **Save rotated image on server** is set, then <*Level0_RotatedImageFormat*> is set to one of the following corresponding values (the first value is the <*Level0_RotatedImageFormat*> value and the second value is the **Rotated image format** selection):

- -1 = Auto
- 0 = TIFF Standard G3 1D
- 1 = TIFF Standard G4
- 2 = JPEG Good
- 3 = TIFF Uncompressed
- 4 = TIFF Standard G3 2D
- 5 = TIFF Packbits
- 6 = TIFF LZW
- 7 = PCX
- 8 = DCX
- 9 = Simple file format
- 10 = Bitmap without compression
- 11 = Bitmap RLE8
- 14 = JPEG Lossless
- 15 = TIFF Group 3 Modified
- 16 = JPEG Min
- 18 = PNG
- 20 = GIFF
- 25 = Adobe PDF Min
- 26 = Adobe PDF Good
- 27 = Adobe PDF Lossless
- 31 = New JPG Compressed TIFF
- 32 = JPEG2000 Lossless
- 33 = JPEG2000 Good

- 34 = JPEG2000 Min
- 35 = JBIG2 Lossless
- 36 = JBIG2 Min
- 38 = Microsoft HD Photo

IA value attributes are:

- **Attributes**: Long, Output
- **Level**: T

<*Level<N>_StartDateTimeUTC***>**
The start date and time.

- **Attributes**: String, Output
- **Level**: 0-7

<*Level<N>_EndDateTimeUTC***>**
The end date and time.

- **Attributes**: String, Output
- **Level**: 0-7

<*NumOutputFile***>**
The number of output files produced while processing the current task. Typically, this will be one or two (assuming errors did not prevent output files from being created). The module will not produce more than two output files per task unless you increase the number of *<OutputFile<n>>* sections in the MDF.

- **Attributes**: Long, Output
- **Level**: T

<*OutputFile1_FullPath***>**
The full path of the current task's *<OutputFile1_OutputFile>* value.

- **Attributes**: String, Output
- **Level**: T

<*OutputFile1_Ext***>**
The file name extension (for example, *PDF*) of the current task's *<OutputFile1_OutputFile>*.

- **Attributes**: String, Output
- **Level**: T

<*OutputFile1_Created*>

An integer that indicates the status of the current task's <*OutputFile1_OutputFile*> value. Valid values are:

- 0 = not created. Generally an output file is not created due to an error in module setup or a run time error while processing tasks.
- 1 = created on server. Corresponds to selecting the **Save to Server** check box in the **Output Format** pane of the **NuanceOCR Setup** window.
- 2 = created on file system. Corresponds to selecting the **Save to File System** check box in the **Output Format**pane of the **NuanceOCR Setup** window.
- 3 = created on both. Corresponds to selecting both check boxes described above.

IA value attributes are:

- **Attributes**: Long, Output
- **Level**: T

<*OutputFile1_ErrorNumber*>

An error code that indicates any error that may have occurred processing the current task's <*OutputFile1_OutputFile*> value.

- **Attributes**: Long, Output
- **Level**: T

<*OutputFile1_ErrorText*>

An error message string that describes the error code in the <*OutputFile1_ErrorNumber*> value.

- **Attributes**: String, Output
- **Level**: T

<*OutputFile2_FullPath*>

The full path of the current task's <*OutputFile2_OutputFile*> value.

- **Attributes**: String, Output
- **Level**: T

<*OutputFile2_Ext*>

The file name extension of the current task's <*OutputFile2_OutputFile*> value.

- **Attributes**: String, Output
- **Level**: T

<*OutputFile2_Created*>

An integer that indicates the status of the current task's <*OutputFile2_OutputFile*> value. Valid values are:

- 0 = Not created. Generally an output file is not created due to an error during module setup or a task processing error in production.
- 1 = Created on the server. Corresponds to selecting the **Save to Server** check box on the **Output Format** pane of the **NuanceOCR Setup** window.
- 2 = Created on the file system. Corresponds to selecting the **Save to File System** check box on the **Output Format** pane of the **NuanceOCR Setup** window.
- 3 = Created on both file system and server. Corresponds to selecting both check boxes described above.

IA value attributes are:

- **Attributes**: Long, Output
- **Level**: T

<*OutputFile2_ErrorNumber*>

An error code that indicates any error that may have occurred processing the current task's <*OutputFile2_OutputFile*> value.

- **Attributes**: Long, Output
- **Level**: T

<*OutputFile2_ErrorText*>

An error message string that describes the error code in the <*OutputFile2_ErrorNumber*> value.

- **Attributes**: String, Output
- **Level**: T

Output status and error values

<*ErrorNumber*>

The error number of the error, if any, that occurred while processing this task.

- **Attributes**: Long, Output
- **Level**: T

<*ErrorText*>

The text string of the error, if any, that occurred while processing this task.

- **Attributes**: String, Output
- **Level**: T

<*ExportResult*>
If all documents contained by the task were exported successfully, this variable is set to zero. Otherwise, this variable contains a negative number indicating the error. Check this variable in a Finish event handler after export to determine whether any documents need to exported again. To determine which documents need to be reexported, check the <*Level0_ErrorNumber*> and <*Level0_ErrorText*> IA values for the node where the error occurred.

- **Attributes**: Long, Output
- **Level**: T

<*Level0_ErrorNumber*>
The error number of the error, if any, that occurred while processing this page.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_ErrorText*>
The text string of the error, if any, that occurred while processing this page.

- **Attributes**: String, Output
- **Level**: 0

<*Level0_Processed*>
Indicates the processing status of the node:

- 1: The node has already been processed, and may be skipped if the module is configured to skip processed nodes on retry after an error.
- 0: The node has not yet been processed.
- **Attributes**: Long, Output
- **Level**: 0

Output statistical variables

<*ReadTime*>
The number of milliseconds the recognition engines spent on the current task from the start of its recognition process. This includes the actual recognition time as well as the time necessary for sorting the output coming from the different recognition modules, spell checking the output, writing the recognition result into the recognition data file, and producing the final output documents.

- **Attributes**: Long, Output
- **Level**: T

<*DecompositionTime*>
The sum of the number of milliseconds the module spent decomposing the page layout for each page in the current task.

- **Attributes**: Long, Output
- **Level**: T

<*PreprocessingTime*>
The sum of the number of milliseconds the engine spent performing the auto-orientation function for each page in the current task.

- **Attributes**: Long, Output
- **Level**: T

<*RecognitionTime*>
The sum of the number of milliseconds all images in the current task spent while each of the recognition engines performed the actual recognition process.

- **Attributes**: Long, Output
- **Level**: T

<*CharacterCount*>
The total number of recognized characters in the current task.

- **Attributes**: Long, Output
- **Level**: T

<*RejectedCharacterCount*>
The total number of unrecognized characters in the current task. This is equivalent to the number of substitute characters inserted by the module as specified in the **Document Recognition** pane.

- **Attributes**: Long, Output
- **Level**: T

<*WordCount*>
The total number of recognized words in the current task.

- **Attributes**: Long, Output
- **Level**: T

<*CorrectedWordCount*>
The total number of words in the current task that were corrected by the Correct Spelling function of the module.

- **Attributes**: Long, Output
- **Level**: T

<*Level0_ReadTime*>
The number of milliseconds the recognition engines spent on the current page from the start of its recognition process. This includes the actual recognition time as well as the time necessary for sorting the output coming from the different recognition modules, spell checking the output, writing the recognition result into the recognition data file, and producing the final output documents.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_DecompositionTime*>
The number of milliseconds the module spent decomposing the page format for the current page.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_PreprocessingTime*>
The sum of the number of milliseconds the engine spent performing the auto-orientation function for the current page.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_RecognitionTime*>
The sum of the number of milliseconds the image of the current page spent while each of the recognition engines performed the actual recognition process.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_CharacterCount*>
The total number of recognized characters in the current page.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_RejectedCharacterCount*>
The total number of unrecognized characters in the current page. This is equivalent to the number of substitute characters inserted by the module as specified in the **Document Recognition** pane.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_WordCount*>
The total number of recognized words in the current page.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_CorrectedWordCount*>
The total number of words in the current page that were corrected by the Correct Spelling function of the module.

- **Attributes**: Long, Output
- **Level**: 0

<*StartTime*>
The time at which processing on the current task commenced. This value is in the format of the operating system's “long time” format.

- **Attributes**: String, Output
- **Level**: T

<*StartDate*>
The date on which processing on the current task commenced. This value is in the format of the operating system's “short date” format.

- **Attributes**: String, Output
- **Level**: T

<*EndTime*>
The time at which processing on the current task completed. This value is in the format of the operating system's “long time” format.

- **Attributes**: String, Output
- **Level**: T

<*EndDate*>
The data on which processing on the current task completed. This value is in the format of the operating system's “short date” format.

- **Attributes**: String, Output
- **Level**: T

<*TotalTime*>
Total processing time of the current task, in milliseconds.

- **Attributes**: Long, Output
- **Level**: T

<*Level0_StartTime*>
The time at which processing on the current page commenced. This value is in the format of the operating system's “long time” format.

- **Attributes**: String, Output
- **Level**: 0

<*Level0_StartDate*>
The date on which processing on the current page commenced. This value is in the format of the operating system's “short date” format.

- **Attributes**: String, Output
- **Level**: 0

<*Level0_EndTime*>
The time at which processing on the current page completed. This value is in the format of the operating system's “long time” format.

- **Attributes**: String, Output
- **Level**: 0

<*Level0_EndDate*>
The date on which processing on the current page completed. This value is in the format of the operating system's “short date” format.

- **Attributes**: String, Output
- **Level**: 0

<*Level0_TotalTime*>
Total processing time of the current page, in milliseconds.

- **Attributes**: Long, Output
- **Level**: 0

<*Operator*>
The user name of the operator who is logged into the module.

- **Attributes**: String, Output
- **Level**: T

Output zone variables

For each zone that has been set up to return recognized text as an IA value, a pair of IA values are populated by the module during production. Up to 50 zones can return IA values without needing to modify the MDF. If more than 50 zones are needed, you can add additional pairs of IA values to the MDF (for example, <*Level0_ZoneValue51_RecText*> and <*Level0_ZoneValue51_TextIsAnsi*>, etc.).

<*Level0_NumZoneValue***>**
The number of zones that were defined in setup mode.

- **Attributes**: Long, Output
- **Level**: 0

<*Level0_ZoneValue<N>_RecText***>**
The recognized text of the Nth zone, where <*N*> corresponds to the number assigned to the zone in the **Zone Setup** window. This value is only populated if the **Return recognized text in IAValue** check box is enabled during setup.

Note: This value can hold recognized text up to the maximum number of characters an IA value can hold. It is possible that full-page zones containing a large amount of text will not entirely fit into an IA value.

- **Attributes**: String, Output
- **Level**: 0

<*Level0_ZoneValue<N>_TextIsAnsi***>**
The format of the text in the IA value <*Level0_ZoneValueN_RecText*>. This IA value will return one of the following Visual Basic True/False values:

- 0 = (FALSE): Code Page. Corresponds to selecting the **As Code Page Characters** on the **IA Values** pane of the **Zone Setup** window.
- -1 (TRUE): ANSI. this corresponds to selecting the **As ANSI Characters** in the **IA Values** pane of the **Zone Setup** window.

IA value attributes are:

- **Attributes**: Long, Output
- **Level**: 0

3.4.3 Output Formats

The following options are available on the **Output Formats** pane.

Output Formats Panel

Adobe® PDF and PDF/A
The *PDF* file contains the recognized characters in the same positions as in the original file. Displaying the generated PDF file in a PDF-reader results in a very similar look to the original document. The text can be searched.

Adobe® PDF and PDF/A edited
This PDF converter does not rely on the position of the recognized characters, so it can be used even after inserting large sections of new text.

Adobe PDF Image Only
This PDF image is not searchable.

Adobe® PDF and PDF/A with image on text
The generated PDF file contains one image for each page in the document and also contains the recognized characters underneath (as "hidden text"). Displaying the generated PDF file in a PDF reader results in a very similar look to the original document. The text can be searched.

Adobe® PDF and PDF/A with image substitutes
Similar to Adobe® PDF, but the problematic recognition cases are handled by the inclusion of smaller image snippets in the output file taken from the original image. Image snippets are also exported for the following cases: words containing suspect characters in the recognized text, words not approved by the checking subsystem (words not found in the dictionary), words containing rejected symbols, and words containing missing symbols.

Excel 2007
Microsoft Office Excel 2007 output format.

Excel 2003, XP
Currently the same as Microsoft Office Excel 97.

HTML 3.2
HTML output. HTML 3.2 is useful to export with **PARTFORMAT** option.

Note: Any image that existed as part of the imported document is not preserved with the output file when this output format is used.

HTML 4.0
HTML output. HTML 4.0 can set the exact position and size of objects. Use this output format with the **FULLFORMAT** option.

Caution
There is a possibility of image data loss when this format is saved to the file system.

Note: Any image that existed as part of the imported document is not preserved with the output file when this output format is used.

InfoPath
A Microsoft Office InfoPath converter that supports the saving of various recognized form elements like check boxes and input lines.

Microsoft Word 2003 (WordML)
A converter for the *XML* based file format of Microsoft Office Word 2003.

PowerPoint 2007
Microsoft Office PowerPoint 2007 output format.

PowerPoint 97 (RTF)
Rich Text Format for Microsoft Office PowerPoint 97.

Publisher 98 (RTF)
Rich Text Format for Microsoft Office Publisher 98.

RTF Word 2000
Rich Text Format for Microsoft Word 2000.

Text - Comma Delimited
Writes recognized text into a tabled text file (Comma delimited text file). A "List Separator" separates the cells and *NL* (new line character) separates the lines of the table.

Text - Formatted
Writes the recognized text into a text file, but tries to retain the format of the page by inserting extra spaces.

Text - Plain
The same as the **Text - Standard** converter, but this inserts line breaks at the end of the lines, instead of only inserting them at the end of the paragraphs.

Text - Standard
Writes the recognized text into a simple text file, which can be read by most text editors and word processors.

Unicode - Comma Delimited
Same as **Text - Comma Delimited**, but uses two byte Unicode characters.

Unicode - Formatted
Same as **Text - Formatted**, but uses two-byte Unicode characters.

Unicode - Standard
Same as **Text - Plain**, but uses two-byte Unicode characters.

Unicode - Stripped
Same as **Text - Standard**, but uses two-byte Unicode characters.

Word 2007
Microsoft Office Word 2007 output format.

WordPad (RTF)
Rich Text Format for Microsoft WordPad.

WordPerfect 9, 10

Corel WordPerfect binary file format for versions 9 and 10.

Note: If a batch was set up in a previous version to use the **WordPerfect 8** output format, then it is automatically changed to use **WordPerfect 10** output format.

XML

Conforms to ScanSoft XML schema (`http://www.scansoft.com/omnipage/xml/ssdoc-schema3.xsd`). It contains almost all layout related information and paragraph and character attributes.

Caution

There is a possibility of image data loss when this format is saved to the file system.

Note: Any image that existed as part of the imported document is not preserved with the output file when this output format is used.

XML Paper Specification (XPS)

An XML-based document format that provides full page layout capabilities, allows digital signatures, and digital rights to be applied to the documents.

3.4.4 Supported and Default Formatting Levels

The different text converter formats use different supported and default formatting level settings. The following table summarizes the available "X" and default "D" settings:

Table 3-1: Text Converter Supported Format Levels

Text Converter Format	No Formatting	Retain Font and Paragraph	Flowing Page	True Page	Spreadsheet
HTML 3.2	X	X			D
HTML 4.0	X	X		D	
InfoPath				D	
Excel 2003, XP	X	X			D
Publisher 98 (RTF)	X	D			
Adobe® PDF and PDF/A				D	

Text Converter Format	No Formatting	Retain Font and Paragraph	Flowing Page	True Page	Spreadsheet
Adobe® PDF and PDF/A edited	X	X		D	
Adobe® PDF and PDF/A with image substitutes, Adobe® PDF and PDF/A with image on text				D	
Text - Formatted, Unicode - Formatted				D	
WordPad (RTF)	X	D			
WordPerfect 9, 10	X	X	X	D	
XML				D	

3.4.5 Supported Languages

NuanceOCR supports the following languages:

Notes

- The recognition engine Machine Print (OMNIFONT_MTX) supports the characters from a limited set of languages. For more information on supported languages, see "Machine Print (OMNIFONT_MTX) Supported Languages" on page 72.
- Hebrew does not support auto-rotation.

Table 3-2: Supported Languages

Afrikaans	Albanian	Aymara	Basque	Bemba	Blackfoot	Brazilian-Portuguese	Breton
Bugotu	Bulgarian (Cyrillic)	Byelorussian (Cyrillic)	Catalan	Chamorro	Chechen	Chinese (Simplified)	Chinese (Traditional)
Corsican	Croatian	Crow	Czech	Danish	Dutch	English	Eskimo (Inuit)

Esperanto	Estonian	Faroese	Fijian	Finnish	French	Frisian	Friulian
Gaelic (Irish)	Gaelic (Scottish)	Galician	Ganda	German	Greek	Guarani	Hani
Hawaiian	Hebrew	Hungarian	Icelandic	Ido	Indonesian	Interlingua	Italian
Japanese	Kabardian	Kasub	Kawa	Kikuyu	Kongo	Korean	Kpelle
Kurdish	Latin	Latvian	Lithuanian	Luba	Lule Sami	Luxembourgian	Macedonian (Cyrillic)
Malagasy	Malay	Malinke	Maltese	Maori	Mayan	Miao	Minankabaw
Mohawk	Moldavian (Cyrillic)	Nahuatl	Northern Sami	Norwegian	Nyanja	Occidental	Ojibway
Papiamento	Pigin English	Polish	Portuguese	Provencal	Quechua	Rhaetic	Romanian
Romany	Ruanda	Rundi	Russian (Cyrillic)	Sami (Lappish)	Samoan	Sardinian	Serbian (Cyrillic)
Serbian (Latin alphabet)	Shona	Sioux	Slovakian	Slovenian	Somali	Sorbian (Wend)	Sotho
Southern Sami	Spanish	Sundanese	Swahili	Swazi	Swedish	Tagalog	Tahitian
Tinpo	Tongan	Tswana (Chuana)	Tun	Turkish	Ukrainian (Cyrillic)	Vietnamese	Visayan
Welsh	Wolof	Xhosa	Zapotec	Zulu			

3.4.6 Machine Print (OMNIFONT_MTX) Supported Languages

The Machine Print (OMNIFONT_MTX) recognition engine supports the following languages or a combination of these languages:

- English
- French
- Spanish
- Italian
- German
- Norwegian
- Portuguese
- Danish

- Dutch
- Finnish
- Swedish
- Brazilian

3.4.7 Spelling Languages

Spelling languages are used during document recognition for automatic spelling correction. The following are the available languages. Additional languages can appear in the module as updates become available.

Table 3-3: Spelling Languages

Catalan	Chamorro	Czech	Danish
Dutch	English	Finnish	French
German	Greek	Hungarian	Italian
Norwegian	Polish	Portuguese	Russian (Cyrillic)
Slovenian	Spanish	Swedish	Turkish

3.4.8 Recognition Engines and Filling Methods

Depending on the type of **Recognition engine** selected, the list of **Filling method** options change. Choosing **No OCR** from the **Filling method** list means that no recognition is attempted. The table indicates the available **Filling method** depending on the **Recognition engine** selected.

Table 3-4: Recognition Engines and Filling Methods

	Machine Print	**9-PIN**	**24-PIN**	*OCR-A*	*OCR-B*	**Magnetic Ink**	**Dot-Digit**	**Dot-Dash**	**Optical Mark**	1D - Barcode	2D - Barcode
Automatic	x	x	x	x	x	x	x	x	x	x	x
Machine Print	x	x	x	x	x						
Multi-Language Machine Print	x		x	x	x						
2-Way voting (PLUS)	x										
3-Way voting (PLUS)	x										
Matrix Matching				x	x	x	x	x			
9-pin Dot Matrix		x									
Barcode module										x	x

	Machine Print	9-PIN	24-PIN	OCR-A	OCR-B	Magnetic Ink	Dot-Digit	Dot-Dash	Optical Mark	1D - Barcode	2D - Barcode
OMR optical mark									x		

3.4.9 Supported Barcode Types

The module supports recognition of the following barcode types:

- *Linear (1D) Barcodes*
  - Codabar
  - Codabar with start/stop character transmission
  - Code 39
  - Code 39 with check digit control
  - Code 39 in full ASCII mode
  - Code 93
  - Code 128
  - EAN 8/13
  - ITF (Interleaved 2 of 5)
  - ITF with check digit control
  - UPC-A
  - UPC-E 6 digit
  - UCC Code 128
- *Matrix (2D) Barcodes*
  - Data matrix
  - PDF417

Chapter 4

Standard OCR

This section includes the Intelligent Capture Standard OCR module: **ecpcore-cob**.

4.1 Introduction

This guide describes the Standard OCR application. It also provides 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.

For more information on the specific capabilities of each client module, see *OpenText Intelligent Capture - Administration Guide (ECPCORE-AON)*.

4.1.1 Understanding the Standard OCR Module

The Standard OCR module performs data extraction from electronic documents and images. For each type of content, the module lets you select an applicable OCR engine processing mode to get better accuracy and productivity.

Also, irrespective of the content type, the Standard OCR module is able to produce an *OCR* file as a result of processing. This file contains OCR data cache which further can be used by other modules:

Module	Specifically	Purpose
Recognition modules	Classification, Extraction	To avoid additional running the full page OCR reading

Module	Specifically	Purpose
Desktop modules	Completion, Identification	When basic rubber banding and/or the **Click to Extract** f

Module	Specifically	Purpose

As an input, the Standard OCR module can process single-page *PDF* and image files. As an output, the module produces PDF or text files. For details, see *Supported File and Image Formats* section of the *Supplemental Reference Guide*.

4.1.2 Electronic Documents Handled by Standard OCR

The following types of electronic documents can be handled by Standard OCR:

- **Text-based PDF documents** (documents that contain textual data only): electronic text is extracted as is.
- **Documents with a mixed content** (documents that contain text, tables, graphical elements, etc.): the full page OCR reading is run. Electronic text is also extracted and the results are used to refine the OCR results.
- **Images**: only the full page OCR reading is run.

4.1.3 Standard OCR Features and Functions

The Standard OCR module performs the following tasks and functions:

- Extracts text from text-based documents, images, or documents with the mixed content by selecting the most effective engine processing mode to process each type of content.
- Creates the OCR data cache file which contains extracted data to be used by other modules in a capture process.
- Converts to other formats. The module can produce searchable PDF and Text files from images.

4.2 Setting Up Standard OCR

Most configuration information for Intelligent Capture client modules is defined during *module setup*. You can run a module in setup mode from Intelligent Capture Designer, from Intelligent Capture Administrator, or from a command prompt with appropriate arguments. For more information, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

When launched in setup mode, a client module displays the **Setup** window in which you can specify the module's configuration settings.

4.2.1 Setting Up a Standard OCR Step

To set up a Standard OCR step:

1. Run the Standard OCR module for setup. For the detailed information on running a module for setup, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.
2. To set up a step, select one of the Standard OCR profiles created in Intelligent Capture Designer and uploaded to the server.

Note: When configuring value assignments between steps in the CaptureFlow, ensure that the level for the Standard OCR input file is set to page level (level 0). By default, the following value assignment is created when adding the **StandardOCR** step to a CaptureFlow `StandardOCR:0.Level0_InputFile`.

You can configure automatic switching between profiles depending on the file content type value generated by Image Converter. To do this, create the following assignment:
`Assign to StandardOCR:0.Profile the value <Profile name> when ImageConverter:0.Level0`

For information on the possible values of <*Content Type*>, see *OpenText Intelligent Capture - Image Handling Modules Guide (ECPCORE-CIH)*.

Related Topics

"Understanding Task Processing in Standard OCR" on page 80.

4.3 Running Standard OCR in Production

After the module has been set up, tested, and placed in a production environment, operators and administrators will run the module in production.

For information related to common production tasks for unattended modules, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

Note: The *Running Modules in Production Mode* section contains production topics that are common to many unattended modules and may not apply to this module.

4.3.1 Understanding Task Processing in Standard OCR

The Standard OCR module processes input documents and images at page level only while it can read data from any level.

When handling electronic documents, the module functions as follows:

- **Trigger level 0**: performs extraction for the page and generates OCR data cache for it.

  Note: If Standard OCR receives a multi-page document at level 0, OCR data cache will be generated for the first page of the document only. In this case, consider to use the **ImageConverter** step prior to the **StandardOCR** step for splitting the multi-page document to pages before sending to Standard OCR. For details on configuring an Image Conversion profile, see *OpenText Intelligent Capture - Designer Guide (ECPCORE-CPD)*.

- **Trigger level 1 and higher**: performs page extraction for all pages under the triggered level and generates OCR data cache for each page.

4.4 Reference—Standard OCR

The topics within this section contain reference information useful while using the application in setup or production.

4.4.1 IA Values

The topics in this section describe IA values that can be used with this application.

Intelligent Capture provides both common and module-specific IA Values. The Common IA Values described in *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)* are valid for all modules.

IA values defined in the Standard OCR module are described in the following table.

Table 4-1: Standard OCR-specific IA Values

IA Value	Trigger Level	Type	Description
<ErrorText>	T	String, Output	A non-localized error message if the task execution fails. Empty if not processed successfully. `Canceled`(English) if the task is canceled after execution.
<Level0_ErrorText>	0	String, Output	A non-localized message of the error, if any, that occurred while processing the specified node.
<Level0_InputFile>	0	File, Input	The input document received for text extraction. If the input document is a multi-page *PDF*, only the first page will be processed.
<OutputFile1>	T	File, Output	The first output PDF or *TXT* file generated from `Level0_InputFile`.
<OutputFile2>	T	File, Output	The second output PDF or *TXT* file generated from `Level0_InputFile`.
<OutputFileExt1>	T	String, Output	Type of the first output PDF or *TXT* file.

IA Value	Trigger Level	Type	Description
<OutputFileExt2>	T	String, Output	Type of the second output PDF or *TXT* file.
<Level0_Resolution>	0	Long, Output	Resolution of the output file which contains the OCR data.
<Machine>	T	String, Output	Name of the machine on which the task is processed.
<OcrDataCache>	0	File, Output	Contains the full text OCR data cache created for the page.
<Profile>	T	String, Input	The **Standard OCR** profile name.
<StartDateTimeUtcC>	T	Date, Output	The date and time at which creating of the batch started, in *UTC* format.
<TotalTimeMilliSec>	T	Long, Output	Total batch processing time, in milliseconds.

4.4.2 Supported Languages

Standard OCR supports the following languages:

- AFRIKAANS
- ALBANIAN
- AZERBAIJANI_CYRILLIC
- AZERBAIJANI_LATIN
- BALTIC
- BASQUE
- BELARUSIAN
- BOSNIAN_LATIN
- BULGARIAN
- CATALAN

  ANDORRA, CATALAN
- CENTRAL_EUROPE
- CHINESE_SIMPLIFIED

- CHINESE_TRADITIONAL
- CHINESE_TRADITIONAL_HK
- CROATIAN
- CYRILLIC
- CZECH_LANGUAGE
- DANISH
- DUTCH

  BELGIUM, NETHERLANDS
- ENGLISH

  AUSTRALIA, CANADA, GREAT_BRITAIN, IRELAND, NEW_ZEALAND, SOUTH_AFRICA, USA
- ESTONIAN
- FAROESE
- FINNISH
- FRENCH

  BELGIUM, CANADA, FRANCE, LUXEMBOURG, SWITZERLAND
- FRISIAN
- GERMAN

  AUSTRIA, BELGIUM, GERMANY, LIECHTENSTEIN, LUXEMBOURG, SWITZERLAND
- GREEK
- GUARANI
- HANI
- HUNGARIAN
- ICELANDIC
- INDONESIAN
- IRISH
- ITALIAN

  ITALY, SWITZERLAND
- JAPANESE
- KAZAKH_CYRILLIC
- KAZAKH_LATIN
- KIRGHIZ_CYRILLIC
- KIRUNDI

- KOREAN
- LATIN
- LATVIAN
- LITHUANIAN
- LUXEMBOURGISH
- MACEDONIAN
- MALAY
- NORWEGIAN
- POLISH
- PORTUGUESE

  BRAZIL, PORTUGAL, SOUTH_AMERICA
- QUECHUA
- RHAETO_ROMANIC
- ROMANIAN
- RUSSIAN

  BELARUS, RUSSIA
- RWANDA
- SERBIAN_CYRILLIC
- SERBIAN_LATIN
- SHONA
- SLOVAK
- SLOVENIAN
- SOMALI
- SORBIAN
- SPANISH

  ANDORRA, ARGENTINA, CENTRAL_AMERICA, CHILE, COLOMBIA, MEXICO, SOUTH_AMERICA, SOUTH_AMERICA_SPANISH, SPAIN, VENEZUELA
- SWAHILI
- SWEDISH
- TAJIK_CYRILLIC
- THAI
- TURKISH
- TURKMEN_CYRILLIC

- TURKMEN_LATIN
- UKRAINIAN
- UZBEK_CYRILLIC
- UZBEK_LATIN
- VIETNAMESE
- WESTERN_EUROPE
- WOLOF
- XHOSA
- ZULU

Chapter 5

Classification Module

This section includes the Intelligent Capture Advanced Recognition Classification module: **ecpcore-ccf**.

5.1 Introduction

After documents have been scanned and enhanced using ScanPlus and Image Processor modules, they can be classified using the Classification module. This module classifies documents automatically by assigning each document to a template defined in a project. Documents that are not classified automatically during the classification phase need to be classified manually using the Identification module. For more information on common operations, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

5.1.1 Classification Projects

The Classification module references a classification project specified during module setup. The classification project defines the set of classification templates that are used to evaluate the images being processed. Each task the module receives during production specifies the classification project that is to be used with that task. The classification project is identified as one of the batch configuration settings, where it can be specified either as a specific file name or as an IA Value. When an IA Value is used to call the classification project, the module can change the classification project on the fly based on the conditions that influence the selected IA Value, such as a barcode on the image or an entry made by a Completion module operator.

When field values are known before classification, their values can be sent to the Classification module using the IA Value <*InputFieldNames*> that contains a list of the field names to be input in classification. Dynamic IA Values can be used to send the value of each input field.

During production, the module loads the classification project specified by the first task it receives, and then it keeps that project in memory either until another task is received that specifies a different classification project or until the module exits. In many cases, a single project can be used for most or all text classification needs. Keeping the classification project loaded makes more efficient use of the machine running the module, increasing its throughput when the same project is needed to process multiple tasks .

5.1.2 Project Templates

A classification project file defines a set of classification templates. Each template models an image class by cataloging a representative sample of the images it should classify and additional information supplied by the person who sets up the project. For example, a given template may be used to classify images of all the types of invoices the company processes. Images that match the invoice template are classified as an "invoice." Multiple templates can be added to further classify invoices according to specific criteria, such as the type of form, a form number, or a specific feature on the invoice.

Each image class must have a corresponding template. Each template will typically contain 10 to 20 model images to compare with the captured images. In addition to the set of model images, additional features can be included to more accurately or thoroughly classify the images.

5.1.3 Classification Algorithms

The classification engine used by the Classification module applies several classification algorithms to the images, including:

- **Full Page Image Analysis**: Evaluates and compares an entire image to the models stored in each template.
- **Handwritten Detection Analysis**: Evaluates images to determine the percentage of handwriting they contain. If higher than a predefined threshold, an image is classified as "handwritten".
- **Full Text Analysis**: Performs *OCR* and evaluates the resulting text for keywords, pattern matches, or regular expressions that were defined in a template.
- **High Precision Anchors**: Selects a feature of an image based on a similar feature that was demarcated on a model image stored in a template.

5.1.4 Classification Requirements

The Classification module requires the following configuration and components to operate successfully:

Includes the module as a step within an IPP

The Classification module interfaces with Intelligent Capture Servers through a process file. To use the module, include it as a step within in *IPP*. The IPP must be compiled into an *IAP* be used in a production environment.

Uses settings defined in setup mode

Before you can run the module within a process in production, you must configure it. To do this, run the module for setup and define specific configuration settings for module within the IPP. You can set up the module at the process level or batch level.

Uses a project file (DPP) defined within Recognition Designer

The *DPP* file is created using the Recognition Designer application and contains the templates, base images, and rules for processing documents. You can use a single DPP file for all document capture jobs or create different project files based on specific processing needs. You specify the DPP file to use during module setup; therefore, each task the module receives can potentially use a different DPP file. During production, the module loads the DPP file into memory and keeps it there until it receives a task that specifies a different project file or until the module exits. The DPP files must be stored in a location that is accessible to all machines that run the module.

Processes tasks in production mode

To process tasks using the module, you must run the Classification module for production. Once the module is started for production, you can choose the mode in which to process tasks.

Includes special error handling instructions to handle runtime errors

When the module completes a task successfully during production, it sets the <*TaskResult*> IA Value to 0. If the module determines an error has occurred while processing a task, the module sets <*ExportResult*> to a non-zero value and logs the error message to the Intelligent Capture Server Event log. When an error occurs, even when one or more of the pages in the task could have produced the error, the module finishes the task with a Success status provided it can process the rest of the pages in the task and the **Skip node that produced the error and continue with remaining nodes** option is selected during module setup. The Success status causes the module to run the `Finish` event so that the step can call its `Finish` event handler. The module fires the `Error` event only if the task finishes with a Failure status. A Failure status can result when the task is aborted, either because of user response to an error prompt or when the **Abort entire task** option is selected during module setup. In this situation, the module should call an `Error` event handler. The `Error` event handler in the IPP should check the exception in the <*ErrorText*> IA Value to determine the cause of the error, and then take appropriate action.

5.2 Setting Up Classification

Most configuration information for Intelligent Capture client modules is defined during *module setup*. You can run a module in setup mode from Intelligent Capture Designer, from Intelligent Capture Administrator, or from a command prompt with appropriate arguments. For more information, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

When launched in setup mode, a client module displays the **Setup** window in which you can specify the module's configuration settings.

5.2.1 Setting Up Classification

Set options on the **Setup** pane in module setup to specify how the Classification module should classify page when it processes tasks in production mode. On this pane, you can:

- Select an existing project file (*DPP*) to use.
- Access Recognition Designer to create a DPP file when none exists.
- Select IA Values within the module *MDF* to make available to Recognition Designer as external values.

To set up Classification:

1. Run the Classification module for setup.
2. Select **Setup** from the navigation panel.
3. In the **Recognition Designer Project** field, specify the full path to and name of an existing project file (DPP) to use to process tasks for this module step. To do this, use one of the following methods:

   - Type the full path and file name of the project into the field. If desired, you can specify one or more IA Values as part of the path that, during production, resolves to the portion of the project path name or location it represents. Also, you can type between `%` the name of an environment variable corresponding to a full or partial path to the project file accordingly: `%variablename%`.

     This requires that you define a new environment variable in the **System Properties** of the machine (**Advanced** > **Environment Variables**). A system restart is required after defining the user or system variable.
   - Click **Browse** and then select **Browse** to display the **Select Recognition Designer Project** window. Navigate to the location of the project you want, highlight it, and click **Open**.
   - Click **Browse** and then click **Insert Value** to display the **Choose Value** window. Select the IA Value you want to use in the **Recognition Designer Project** field. Repeat this step to insert additional IA Values. When you insert an IA Value in this way, the module prepends an "@" character and encloses the value in parentheses, to distinguish it from ordinary character data in the field. For example:

     <@(Scan.Level_0_KeyEntry_2)>

     Note: The path and file name you specify must be accessible to all machines that will run this module. The easiest way to do this is to store the projects in a shared network directory and specify the path name as a *UNC* path.
4. When no project file exists, click the **Recognition Designer** button to run Recognition Designer where you can create a project with the desired

configuration. Remember to save the DPP file to a shared location accessible to all machines that will run this module step.

Note: For best results, define templates in for the project using images that have the same resolution as the production images. For example, if you are scanning images with a resolution of 200 *DPI*, the templates should use images that have a resolution of 200 *DPI* as well.

5. Select the **Process backside images as pages** option from the **Production Settings** if you want backside pages to be processed as individual pages.

   If you select this option in Classification, we strongly recommend you select it in the three other instances, otherwise this option may not work properly. For example, if you clear this option in Classification, only frontside images are classified. Since backside images have not been classified, those pages are not processed and thus ignored (no associated templates, index family, recognition or control script).

   Note: The **Process backside images as a page** option does not have an effect on how pages are counted when using Dispatcher Manager. This option applies to Classification module only and does not apply to Dispatcher Manager standalone.

6. The tables beneath **External IA Values** enable you to link IA Values within Intelligent Capture to external values within Recognition Designer. From the **Levels** list box, select a level to display the IA Values available for that level within this module. Only External IA Values from the current MDF for the module can display in the list. Options are:

   - **Level 0 IA Values**: Select to display all of the Level 0 (page level) IA Values available for this module.
   - **Level 7 IA Values**: Select to display all of the Level 7 (batch level) IA Values available for this module.

7. From the **Available** table, select the specific IA Values to make accessible to Recognition Designer as external values, and add them to the **Selected** table. To sort the external values list, select the **Name**, **Level** or **Type** column heading.

8. Do one of the following:

   1. Click **Apply** to save the settings and continue with setup.
   2. Click **OK** to save new settings and close the setup window.

5.2.2 Setting Up Error Handling

Define options for handling production errors during module setup.

Note: When a module is run as a service, error handling options defined during setup are ignored.

To define error handing:

1. Select **Error** from the navigation panel during module setup to set options that specify how the module should respond when an error occurs during processing. Select from the following options to define how the system responds when and error occurs:

   - **When an error occurs**: Contains options that determine how the module handles errors encountered by during production in attended mode (an operator is monitoring the machine where the module is running).

     – **Prompt for an action** displays a prompt to the operator when an error occurs during production. Suspend processing until the operator responds to the error.

     – **Automatically respond with the following options:** automates error handling during production when the module is run as an unattended process (i.e., a service). No operator prompts display during production. With this option selected, select from the following options:

       ○ **Abort entire task**: Aborts the current task and returns an appropriate <*ExportResult*> IA Value, which can be evaluated by instructions in the process `Finish` event handler.

       ○ **Skip node that produced the error and continue with remaining nodes**: Returns an appropriate <*ExportResult*> IA Value for the node in which the error occurred which is evaluated by instructions in the process `Finish` event handler.

       ○ **Stop receiving tasks after current task**: Equivalent to the operator choosing the `Stop` command. When not selected, and an error occurs, the module will continue to accept tasks from the Intelligent Capture Server.

       ○ **Set the batch priority to 0**: Takes the batch offline and prevents further processing by any module until the administrator resets the batch priority. When this option is not selected and an error occurs, the batch priority remains unchanged in response to the <*ExportResult*> IA Value, except as specified in the process `Error` or `Finish` event handlers.

       ○ **Set batch error**: Sets the batch to an error status and prevents further processing by the module until the administrator clears the error.

2. Select from the following recovery options, to determine how the module recovers from production errors:

- **Recovery options:** Contains options that determine how the module recovers from errors during production:
  - **Automatically retry**: To repeat the operation that caused the error. Before the error is reported, the operation will be repeated based on the **Number of times to retry before reporting error**.
  - **Skip nodes that have already been successfully processed by this instance**: Prevents the module from reprocessing nodes that have already been successfully processed. When this field is cleared, the module will reprocess all nodes in the task, even those that have already been successfully processed.

5.2.3 Setting Up Statistics

Select **Statistics** from the navigation panel during module setup to set statistics collection options for the module step within the process.

To set up statistics collection:

1. Select **Send statistical data to IA values** to capture statistics in the form of IA Values defined within the *MDF* file for the module. Users can view this data from the Intelligent Capture Administrator and export the values to a third-party repository using an export module.

   Note: Create a step in the process for the exporter module to successfully export the IA Values defined within the `DPStatistics.mdf`.

2. Select **Send statistical data to IA file** to capture statistics to an *XML* stage file for each task. The XML stage file is generated and stored into the <*StatsXMLFile*> IA Value at the task level. Use the Standard Export module to export this XML stage file to a file system.

   Note: Create and run a batch using the process that contains this module step and then find the XML file stored in the <*StatsXMLFile*> IA Value at the task level.

3. Select **Send statistical data to database** to capture statistics to a SQL Server database created specifically to collect statistics. Use the Statistics module to connect to this database and analyze the results.

4. To test the connection to the database specified in the **Connection string** field, select **Test connection** .

5.3 Running Classification in Production

After the module has been set up, tested, and placed in a production environment, operators and administrators will run the module in production.

For information related to common production tasks for unattended modules, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*. For information on production topics that are common to unattended modules and may not apply to this module, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

Note: You can use the Advanced Cloud OCR engine only when you enable classification with information extraction-based projects. To use the Advanced Cloud OCR engine, you require a valid credential file from Google with the following content:

```
{ "type": "", "project_id": "", "private_key_id": "", "private_key": "",
"client_email": "", "client_id": "", "auth_uri": "", "token_uri": "",
"auth_provider_x509_cert_url": "", "client_x509_cert_url": "" }
```

The plugin uses the environment variable *<GOOGLE_APPLICATION_CREDENTIALS>* to get the credentials file. If the environment variable is not set or is empty, the plugin uses the `GoogleConfig.json` file located in the `IEE2\Config` folder.

5.3.1 Understanding Task Processing in Classification

When ready to process documents, run each of the modules that are included as steps in the process for production, specifying for each module whether to accept tasks from all available batches, or only accept tasks from a specific batch.

Depending on workload and licensing arrangements:

Each module can run on its own dedicated machine.
For example, the ScanPlus, Image Processor, Completion, Classification, Identification, Extraction, Completion and custom export modules can each be run on separate machines. The ScanPlus, Completion, Identification, and Completion modules are attended modules and require operator interaction; the other modules do not. Unattended modules can be located in a server room or other controlled-access location and, when properly configured, can generally be left unattended for long periods of time.

Several different modules can run on a single machine.
For example, the ScanPlus and Classification modules can each run on their own dedicated machines, but Image Processor, Completion, and custom export modules can be run on a single machine. Because only the Completion module requires ongoing operator attention, the other modules do not interfere with the flow of work through the steps in the process (assuming the volume of pages being processed does not overwhelm the capacity of the machine).

The same module can run on multiple machines.

In cases where throughput needs to be high and one or more modules creates a bottleneck, a module can be run on multiple machines as a way of distributing the load. Depending on the complexity and size of the projects, Classification can take considerably longer to process a page than other modules, resulting in a backlog of work.

Regardless of how the Intelligent Capture system is configured, each running module accepts tasks as its workload allows and as they become available on the Intelligent Capture Server.

To use the Classification module during production, start the module in production mode on each machine where the module will be processing tasks, connect to and log onto one or more Intelligent Capture Servers, select a processing mode, and handle any errors that occur. Because the module is designed to be run unattended, many users will want to streamline these steps by specifying command line parameters that start the module, log onto Intelligent Capture Servers, and start processing in Waiting for a Task mode in a single step (after having set up error handling to silently log errors to the Intelligent Capture Server).

5.3.1.1 Processing Tasks in Attended Mode

In attended mode, an operator processes tasks for Classification.

To process tasks:

1. Run Classification for production. The **Action** panel with options for running and processing batches.

   - **Action** contains options for running batches.
   - **Task Information** displays task and batch information.
   - **Detected errors** displays error messages that occur during production.
   - **Processing log** displays the date and time of events, node numbers, batch name, task progress, warnings, and error messages.

2. Respond to **Errors** reported during production.

   - ***Acknowledge Single Error***: Report single errors as they occur.
   - ***Acknowledge All Errors***: Report all errors.

   The **Filter** option controls the error reporting options displayed.

3. Select the **Export log to file** option to display the **Save As** window to specify a location for the `Dispatcher Classification.log` file. This file contains the information specified from the **Filter** button.

4. Continue processing batches until all tasks are complete.

5.3.2 Understanding Errors Detected in Production

When monitoring the progress of each task and an error occurs in production, the error log code displays in the **Detected errors** pane.

For a complete list of client module error and log codes, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

5.4 Reference—Classification

The topics within this section contain reference information useful while using the application in setup or production.

5.4.1 Input IA Values

Input IA Values include input file variables, which serve as pointers to stage files, and input processing variables, which store setup data that a module uses to process tasks, including default and user-defined module settings.

Table 5-1: Input IA Values for Classification

IA Value	Description
<InputImage>	The input image of the task on the Intelligent Capture Server. The file that corresponds with *<InputImage>* is a *TIFF* file in the color and compression format output by the preceding module. This is the trigger value of the module, which means that after it is set to a non-zero value, the Intelligent Capture Server sends tasks to a running validation machine. Completion accepts tasks at any level (0-7). This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Static, File, Input, Trigger • **Level**: 0
<InputFieldNames>	List of the index field names passed to the module. Values are delimited by commas: *<<Field1>, <Field2>>*). This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Static, String, Input • **Level**: 0

IA Value	Description
<InputXMLData>	Contains the Recognition Designer *XML* fragment before Recognition Designer *API* call. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Static, String, Input • **Level**: 0
<Level<n>_Processed>	A value that is set to 1 after each level n node (0,7) in the task has been processed. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Long, Input, NoTrigger • **Level**: *<n>*
<OcrEngine>	Select the OCR engine for the classification request. The accepted values are ***GOOGLE*** and ***CRE***. You can use these OCR engines only when you enable classification with information extraction-based projects. The default value is ***CRE***. • **Level**: *<0>* • **Type**: String

5.4.2 Output IA Values

Output IA Values include output processing variables, which store data that was generated by the module during processing. Some of the following output values are created dynamically and do not need to be specified in the *MDF*.

Table 5-2: Output IA Values for Classification

IA Value	Description
<NeedClassificationEdit>	True if an image of the task must be manually classified for advanced recognition (AR) and if a document must be classified for Information Extraction (IE). • **Attributes**: Static, Boolean, Output • **Level**: T **Note:** If **Pre-Index** is enabled in the classification configuration, and a field is not found, then the *<NeedClassificationEdit>* flag is set to **True**. This will require manual confirmation to complete the classification.
<StatsXMLFile>	Statistics *XML* file. This file is generated only when Statistics is enabled and this option is selected in setup mode. • **Attributes**: Static, File, Output • **Level**: T
<TemplateCount>	The total number of processed template objects (*<DIATemplate>*). If the project contains more than ten templates predefined within the MDF, this definition will have to be expanded. • **Attributes**: Static, Long, Output • **Level**: T
<Template_<n>>	The *<DIATemplate>* object containing statistical data of a processed template, where n is the number of the template. Currently, a maximum of ten templates are supported within a single project. • **Attributes**: Static, DIATemplate, Output • **Level**: T

IA Value	Description
<TaskResult>	When all documents contained by the task are exported successfully, this variable is set to zero. Otherwise, this variable contains a non-zero number indicating that an error occurred. You can check this variable in a `Finish` event handler after export to determine whether any documents need to exported again. To determine which documents need to be re-exported, check the *<Level<n>_ErrorText>* IA Value for the node where the error occurred. • **Attributes**: Long, Output, Trigger • **Level**: T
<ErrorNumber>	The module sets this value to 0 if no error occurred, and to a non-zero value if there was an error. To determine the error that occurred, you must check the exception portion of the *<ErrorText>* value. • **Attributes**: Long, Output, Trigger • **Level**: T
<ErrorText>	The text string of the error, if any, that occurred while processing this task. • **Attributes**: String, Output, Trigger • **Level**: T
<ErrorName>	The text string name of the error, if any, that occurred while processing this task. • **Attributes**: String, Output, Trigger • **Level**: T
<StartDate>	The date on which processing on the current task commenced. This value is in the format of the operating system's “short date” format. • **Attributes**: String, Output, Trigger • **Level**: T
<StartTime>	The time at which processing on the current task commenced. This value is in the format of the operating system's “long time” format. This value is in milliseconds. • **Attributes**: String, Output, Trigger • **Level**: T

IA Value	Description
<EndDate>	The date on which processing on the current task completed. This value is in the format of the operating system's "short date" format. • **Attributes**: String, Output, Trigger • **Level**: T
<EndTime>	The time at which processing on the current task completed. This value is in the format of the operating system's "long time" format. This value is in milliseconds. • **Attributes**: String, Output, Trigger • **Level**: T
<TotalTime>	Total processing time of the current task, in milliseconds. • **Attributes**: Long, Output, Trigger • **Level**: T
<Operator>	The user name of the operator who is logged into the module. • **Attributes**: String, Output, Trigger • **Level**: T
<Level<n>_Processed>	A value that is set to 1 after each level *<n>* node (0,7) in the task has been processed, where *<n>* represents the current processing level. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Long, Input, Output, NoTrigger • **Level**: *<n>*
<Level<n>_ErrorNumber>	The module sets this level *<n>* value to 0 if no error occurred and to a non-zero value if there was an error, where *<n>* represents the current processing level. To determine the error that occurred, you must check the exception portion of the *<Level<n>_ErrorText>* value. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Long, Output • **Level**: *<n>*

IA Value	Description
<Level<n>_ErrorText>	The text string of the error, if any, that occurred while processing this page, where *<n>* represents the current processing level. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_ErrorName>	The text string name of the error, if any, that occurred while processing this page, where *<n>* represents the current processing level. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_StartDate>	The date on which processing on the current page commenced, where *<n>* represents the current processing level. This value is in the format of the operating system's “short date” format. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_StartTime>	The time at which processing on the current page commenced, where *<n>* represents the current processing level. This value is in the format of the operating system's “long time” format. This value is in milliseconds. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_EndDate>	The date on which processing on the current page completed, where *<n>* represents the current processing level. This value is in the format of the operating system's “short date” format. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: String, Output • **Level**: *<n>*

IA Value	Description
<Level<n>_EndTime>	The time at which processing on the current page completed, where *<n>* represents the current processing level. This value is in the format of the operating system's "long time" format. This value is in milliseconds. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_TotalTime>	Total processing time of the current page, in milliseconds, where *<n>* represents the current processing level. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Long, Output • **Level**: *<n>*
<OutputImage>	Output image returned by modules. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Static, File, Output • **Level**: 0
<OcrDataCache>	Contains the full text *OCR* data cache. It enables Identification and Collector to use these results in the workflow. • **Attributes**: Static, File, Input • **Level**: 0
<OutputXMLData>	Contains the Recognition Designer XML fragment after Recognition Designer *API* call. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Static, String, Output • **Level**: 0
<TemplateCode>	Contains the template code setup within the project. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Static, String, Output • **Level**: 0

IA Value	Description
<Level0_NeedClassificationEdit>	True (1) if this page needs to be manually classified. This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Static, Boolean, Output • **Level**: 0
<OutputFieldNames>	List of the index field names returned by the module that correspond to the pre-index fields defined within the project file (*DPP*). Values are delimited by commas: *<<Field1>, <Field2>>*). This IA Value permits access to an external value in Recognition Designer. • **Attributes**: Static, String, Output • **Level**: 0
<ImageOutputFileExtension>	Defines an extension for stored image files (such as .jpg or .bmp). • **Attributes**: String

Chapter 6

Collector Module

This section includes the Intelligent Capture Advanced Recognition Collector module: **ecpcore-cco**.

6.1 Introduction

Collector is a component of the Production Auto-Learning (*PAL*) feature that collects documents during production. These documents are stored in a defined location called **documents storage**. PAL includes the Collector module, a **Supervisor** for managing learning of collected documents, and the **document storage** folder.

Before documents can be collected, Collector must be added as a step in the CaptureFlow (*XPP*), usually after the validation step. Documents must be processed to retrieve the document type and the field values.

Before using the module, Collector must be enabled in the recognition project in the **Project Options** window of Recognition Designer. To be collected, a document must be assigned to a generic or a text matching template. During development, the recognition project designer selects templates in the **Project Options** window of Recognition Designer. The selected templates define which unclassified documents will be collected in production. During production, Collector analyses each document, selects only the documents classified with generic or text matching templates and sends them to document storage. Instructions to select collectable templates can be found in *OpenText Intelligent Capture - Recognition Designer Guide (ECPCORE-CRC)*.

When a document is collected, Collector copies the information required for automatic learning from the Intelligent Capture Server to the **document storage** folder. Information copied includes the following files for each document page:

- The image file:
  - `.TIF`
  - single-page PDF (`*.PDF`)
- The `OCR` file, including the OCR data cache generated by Standard OCR, that retrieves all the data read by the recognition engine
- The `XML` file that retrieves the documents' type and the fields' value

Note: The OCR file is not copied to the document storage if the Collector *OCR* cache is empty. For example, this may happen if the recognition engine was not triggered for a document during classification and recognition. This also happens when classification and recognition steps are not configured properly to route their OCR output to the Collector OCR cache. In that case, document

learning is incomplete or cannot be performed at all. Instructions to configure data transition between steps can be found in topic "IA Value Assignments: Example" on page 119.

Collected documents are regularly checked by the **Supervisor**, which evaluates the documents, initiates learning, and creates new standard (graphical) and textual templates. The recognition project is then updated to include these new templates. Instructions to create new templates can be found in *OpenText Intelligent Capture - Recognition Designer Guide (ECPCORE-CRC)*.

Finally, the documents used to create templates are purged by the **Supervisor** from **document storage** to avoid processing them again the next time the automatic learning runs. Instructions to purge the document storage can be found in *OpenText Intelligent Capture - Recognition Designer Guide (ECPCORE-CRC)*.

Install the Collector module, the **Supervisor** and the **document storage** on the same machine. This enables faster image comparison, project updates, and better performance. Installing these components on one machine is also more secure since it avoids using a *UNC* path, which would be required for documents stored on an external server.

6.1.1 Comparing Advanced Recognition and Information Extraction Learning

Some key differences between advanced recognition (AR) and information extraction (IE) learning are as follows:

Table 6-1: Comparison of AR and IE learning

Advanced recognition	Information extraction
Collects files for later learning by the supervisor	Learns on the spot as each document is processed
Runs at level 0 (or above) because each page is learned separately	Runs at level 1 (or above) because each document is learned separately

6.2 Setting Up Collector

Most configuration information for Intelligent Capture client modules is defined during *module setup*. You can run a module in setup mode from Intelligent Capture Designer, from Intelligent Capture Administrator, or from a command prompt with appropriate arguments. For more information, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

When launched in setup mode, a client module displays the **Setup** window in which you can specify the module's configuration settings.

6.2.1 Setting Up Collector

Configure options in the module setup to specify how the Collector module should recognize pages when it processes tasks in production mode. You can:

- Select an existing project file (*DPP*) to use.
- Process backside images as individual pages.

To set up Collector:

1. Run the Collector module for setup.
2. To choose a recognition project, select the **Select Recognition Project** option and choose a project from the dropdown list.
3. To choose a recognition project from an IA value, select the **Insert value** option and click the **Browse** button to select an IA value.

   Alternatively, enter the formatted IA value that will contain the project path (or name) in the format string specifying the recognition project path. Repeat to insert all required IA values. When you insert an IA value in the field, the module prepends an "at" symbol (@) and encloses the value in parentheses to distinguish it from ordinary character data in the field. For example:

   ```
   <@(Scan.Level_0_KeyEntry_2)>
   ```

4. If using an Advanced Recognition project, in the Production Settings area, select the **Process backside images as pages** option if you want backside pages to be collected and processed as individual pages. Otherwise, only frontside pages will be collected.

   Note: If this option is selected in Collector setup settings but not selected for other recognition steps included in a CaptureFlow, then an error is generated because the <*InputXMLData*> IA value is empty. If you want this option to be correctly processed, then select it in all recognition steps in the collector step in the CaptureFlow.

5. If using an Information Extraction project, from the **Learning Mode** list, select one of the following options:

   - **Classification**
   - **Extraction**
   - **Both**
   - **None**

   Note: When selected, the **None** option is used to remove temporary data generated by Information Extraction. That data will clean itself up eventually, but using the Collector with **None** is more efficient.

6. To save the configuration settings, click **OK**.

6.2.2 Setting Up Error Handling

Define the error handling options to specify how Collector should respond when an error occurs during processing.

To set up error handling:

1. Select **Error** from the navigation panel of the module setup window. In the **Error Handling Settings** tab, select from the following options to determine how Collector handles errors encountered during production when run as an application:

Table 6-2: Collector Error Handling in Attended Mode

Parameter/Option	Description
Prompt for an action	Displays a prompt to the operator when an error occurs during production. Suspends processing until the operator responds to the error.
Automatically respond with the following options:	Automates error handling during production, with no prompts displayed to the operator. Select one of the following options:
Abort entire task	Aborts the current task and returns an appropriate <*ExportResult*> IA value, which can be evaluated by instructions in the process `Finish` event handler.
Skip node that produced the error and continue with remaining nodes	Returns an appropriate <*ExportResult*> IA value for the node in which the error occurred which is evaluated by instructions in the process `Finish` event handler.
Additionally, select the following options:	

Parameter/Option	Description
Stop receiving tasks after current task	Equivalent to the operator choosing the `Stop` command. When not selected, and an error occurs, the module will continue to accept tasks from the Intelligent Capture Server.
Set the batch priority to 0	Takes the batch offline and prevents further processing by any module until the administrator resets the batch priority. When this option is not selected and an error occurs, the batch priority remains unchanged in response to the *<ExportResult>* IA value, except as specified in the process `Error` or `Finish` event handlers.
Set batch error	Sets the batch to an error status and prevents further processing by the module until the administrator clears the error.

Note: When the module is run as a service, the above error handling options are ignored. The module skips the node that produced the error, continues to receiving tasks, and does not set the batch priority to 0 unless the service is stopped or paused.

2. Select from the following recovery options to determine how the module recovers from production errors:

   - **Recovery options:** Contains options that determine how the module recovers from errors during production:

     – **Automatically retry**: To repeat the operation that caused the error. Before the error is reported, the operation will be repeated based on the **Number of times to retry before reporting an error**.

     – **Skip nodes that have already been successfully processed by this step**: Prevents the module from reprocessing nodes that have already been successfully processed. When this field is cleared, the module will reprocess all nodes in the task, even those that have already been successfully processed.

6.3 Running Collector in Production

After the module has been set up, tested, and placed in a production environment, operators and administrators will run the module in production.

For information related to common production tasks for unattended modules, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

Note: The *Running Modules in Production Mode* section contains production topics that are common to many unattended modules and may not apply to this module.

6.3.1 Understanding Task Processing

When you are ready to process documents, run each of the modules that were included as steps in the process for production. For each module, specify whether to accept tasks from all available batches, or only accept tasks from a specific batch.

Depending on workload and licensing arrangements:

Each module can run on its own dedicated workstation
For example, the ScanPlus, Image Processor, Completion, Classification, Identification, Collector and custom export modules can each be run on separate workstations. The ScanPlus, Completion, and Identification modules are attended modules and require operator interaction; the other modules do not. Unattended modules can be located in a server room or other controlled-access location and, when properly configured, can generally be left unattended for long periods of time.

Multiple different modules can run on a single workstation
For example, the ScanPlus and Classification modules can each run on their own dedicated workstations, but Image Processor, Completion, and custom export modules can be run on a single workstation. Because only the Completion module requires ongoing operator attention, the other modules do not interfere with the flow of work through the steps in the process (assuming the volume of pages being processed does not overwhelm the capacity of the workstation).

The same module can run on multiple workstations
In cases where throughput needs to be high and one or more modules creates a bottleneck, a module can be run on multiple workstations as a way of distributing the load. Depending on the complexity and size of the projects, Classification can take considerably longer to process a page than other modules, resulting in a backlog of work.

Regardless of how the system is configured, each running module accepts tasks as its workload allows and as they become available on the Intelligent Capture Server.

To use the Collector module during production, you must (on each workstation where the module will be processing tasks) start the module in production mode, connect to and log onto one or more Intelligent Capture Servers, select a processing

mode, and handle any errors that occur. Because the module is designed to be run unattended, many users will want to streamline these steps by specifying command line parameters that start the module, log onto Intelligent Capture Servers, and start processing in Waiting for a Task mode in a single step (after having set up error handling to silently log errors to the Intelligent Capture Server).

6.3.2 Understanding Errors Detected in Production

When monitoring the progress of each task and an error occurs in production, the error log code displays in the **Detected errors** pane.

For a complete list of client module error and log codes, see *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

6.4 Reference—Collector

The topics within this section contain reference information useful while using the application in setup or production.

6.4.1 IA Values

The topics in this section describe IA values that can be used with this application.

The information in this section describes IA values declared in the *MDF* file of Collector. The information about non-MDF IA values and MDF values common for every Intelligent Capture module can be found in *OpenText Intelligent Capture - Module Reference (ECPCORE-CMD)*.

6.4.1.1 Input IA Values

The following table summarizes the information about input IA values defined in the module *MDF*.

Table 6-3: Input IA Values for Collector

IA Value	Description
<InputImage>	The input image or single-page PDF of the task on the Intelligent Capture Server. References a single-page PDF or `.TIF` file in the color and compression format output as specified in the preceding step in a CaptureFlow. This is the trigger value, which means that after it is set to a non-zero value, the Intelligent Capture Server sends a task to a running Collector workstation. Collector accepts tasks at any level (0-7). • **Attributes**: Static, File, Input • **Level**: 0

<ImageResolution>	Resolution at which *<InputImage>* was scanned. This IA value must be routed from ScanPlus, Standard OCR, or Image Converter directly to Collector; otherwise, the image resolution is incorrect and images are not collected.
	
	Image resolution can also come from Standard Import when batches are created using the Intelligent Capture Web Client or the distributed Web Client.
	
	The resolution should be set to one of the following:
	• For PDF and image files with images or graphical elements, use the image resolution from ScanPlus or Image Converter. For example:
	`ScanPlus:0:ImageResolution`
	• For Standard OCR-processed PDF and image files that also have their OCR data cache used as the input OCR file, use
	`StandardOCR:0.Level0_Resolution`
	.
	• For text-only PDF files, use the Recognition project resolution.
	• For a Recognition project with an undefined resolution, use 300.

IA Value	Description
<InputXMLData>	A reference of the `XML` file that Collector copies to the document storage folder for learning. Input *XML* data includes the document type and values of a particular document page. In a typical CaptureFlow configuration, the `XML` file per page is created in classification and routed to recognition (to include the extracted values), and, optionally, to validation (to include user input in the index fields). Any of these steps can be configured to route XML output to Collector's *<InputXMLData>* value. If input XML data requires pre-*PAL* processing, Collector can implement scripting. A reference of the resulting `XML` file is stored in the *<OutputXMLData>* value of Collector, in which case Collector copies that modified file to the document storage folder for learning. • **Attributes**: Static, String, Input • **Level**: 0
<Level<n>_Processed>	A value that is set to 1 after each level *<n>* (0,7) node in the task has been processed, where *<n>* represents the current processing level. • **Attributes**: Long, Input, Output, NoTrigger • **Level**: *<n>*
<UimData>	*<UimData>* contains all of the data from the document and information about where on the document the data is located. • **Level**: 1 This setting is necessary for Information Extraction learning, but not for Advanced Recognition learning.
<ClassificationID>	*<ClassificationID>* is identification tag created from the Classification Module that assists IEE in classifying documents. • **Level**: 0 This setting is necessary for Information Extraction learning, but not for Advanced Recognition learning.

6.4.1.2 Output IA Values

The following table summarizes the information about output IA values that store data generated by Collector during processing. Some of the following output values are created dynamically and do not need to be specified in the *MDF*.

Table 6-4: Output IA Values for Collector

IA Value	Description
<TaskResult>	When all documents contained by the task are exported successfully, this variable is set to zero. Otherwise, this variable contains a non-zero number indicating that an error occurred. You can check this variable in a `Finish` event handler after export to determine whether any documents need to be exported again. To determine which documents need to be re-exported, check the *<Level><n>_ErrorText>* IA value for the node where the error occurred. • **Attributes**: Long, Output, Trigger • **Level**: T
<ErrorNumber>	The module sets this value to 0 if no error occurred, and to a non-zero value if there was an error. To determine the error that occurred, you must check the exception portion of the *<ErrorText>* value. • **Attributes**: Long, Output, Trigger • **Level**: T
<ErrorText>	The text string of the error, if any, that occurred while processing this task. • **Attributes**: String, Output, Trigger • **Level**: T
<ErrorName>	The text string name of the error, if any, that occurred while processing this task. • **Attributes**: String, Output, Trigger • **Level**: T
<StartDate>	The date on which processing on the current task commenced. This value is in the format of the operating system's “short date” format. • **Attributes**: String, Output, Trigger • **Level**: T

IA Value	Description
<StartTime>	The time at which processing on the current task commenced. This value is in the format of the operating system's "long time" format. This value is in milliseconds. • **Attributes**: String, Output, Trigger • **Level**: T
<EndDate>	The date on which processing on the current task completed. This value is in the format of the operating system's "short date" format. • **Attributes**: String, Output, Trigger • **Level**: T
<EndTime>	The time at which processing on the current task completed. This value is in the format of the operating system's "long time" format. This value is in milliseconds. • **Attributes**: String, Output, Trigger • **Level**: T
<TotalTime>	Total processing time of the current task, in milliseconds. • **Attributes**: Long, Output, Trigger • **Level**: T
<Operator>	The user name of the operator who is logged into the module. • **Attributes**: String, Output, Trigger • **Level**: T
<Level<n>_Processed>	A value that is set to 1 after each level *<n>* (0,7) node in the task has been processed, where *<n>* represents the current processing level. This value can be accessed by other CaptureFlow steps. • **Attributes**: Long, Input, Output, NoTrigger • **Level**: *<n>*

IA Value	Description
<Level<n>_ErrorNumber>	The module sets this value to 0 if no error occurred, and to a non-zero value if there was an error, where *<n>* represents the current processing level. To determine the error that occurred, you must check the exception portion of the *<Level<n>_ErrorText>* value. • **Attributes**: Long, Output • **Level**: *<n>*
<Level<n>_ErrorText>	The text string of the error, if any, that occurred while processing this page, where *<n>* represents the current processing level. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_ErrorName>	The text string name of the error, if any, that occurred while processing this page, where *<n>* represents the current processing level. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_StartDate>	The date on which processing on the current page commenced, where *<n>* represents the current processing level. This value is in the format of the operating system's “short date” format. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_StartTime>	The time at which processing on the current page commenced, where *<n>* represents the current processing level. This value is in the format of the operating system's “long time” format. This value is in milliseconds. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_EndDate>	The date on which processing on the current page completed, where *<n>* represents the current processing level. This value is in the format of the operating system's “short date” format. • **Attributes**: String, Output • **Level**: *<n>*

IA Value	Description
<Level<n>_EndTime>	The time at which processing on the current page completed, where *<n>* represents the current processing level. This value is in the format of the operating system's "long time" format. This value is in milliseconds. • **Attributes**: String, Output • **Level**: *<n>*
<Level<n>_TotalTime>	Total processing time of the current page, in milliseconds, where *<n>* represents the current processing level. • **Attributes**: Long, Output • **Level**: *<n>*
<OutputImage>	Output image. The value references the image file (`.TIF`) that Collector copies to document storage folder for learning. • **Attributes**: Static, File, Output • **Level**: 0
<OutputXMLData>	Output *XML* data with the document type and values of a particular document page. A step that executes prior to Collector, such as classification, recognition, or validation, can route its XML output directly to this value rather than to *<InputXMLData>*. If XML data is routed to *<InputXMLData>* and then modified by scripting, Collector puts the reference of the resulting `XML` file to *<OutputXMLData>*. Collector copies the final `XML` file to the document storage for learning. • **Attributes**: Static, String, Output • **Level**: 0

IA Value	Description
<OCRDataCache>	Contains the *OCR* data cache that Collector copies to document storage folder for learning. This IA value must be routed from classification, recognition, or Standard OCR directly to Collector. If this value is not assigned, Collector cannot get OCR output from any of these steps and considers its OCR cache empty. As a result, Collector does not put the `OCR` file to the storage folder, and document page learning cannot be complete or does not perform at all. • **Attributes**: Static, File, Input • **Level**: 0 For a single-page PDF file generated by Standard OCR, the OCR data cache from the Standard OCR module. For example: `StandardOCR:0.OcrDataCache`
<ImageOutputFileExtension>	(Default is `.TIF`.) Defines the extension for images processed in a batch. This output value is routed from Standard Import, ScanPlus or Image Converter. To process *JPEG* images in a project, the output file format option must be set to JPEG in the ScanPlus settings. • **Attributes**: String For a single-page PDF file, the value must be one of the following: • `pdf` • The output file type from Standard Import, ScanPlus or Image Converter. For example: `StandardImport:1.OutputFileType`

6.4.2 IA Value Assignments: Example

The role of Collector in a CaptureFlow consists in passing to the document storage (a PAL component) a set of files for each document page. These files are then used by the PAL when learning the document. These files include: the image file (`.TIF`) or single-page PDF (`*.PDF`), the `OCR` file with data retrieved by the recognition engine or an OCR data cache file generated by Standard OCR, and the `XML` file with the document type and the field values. These files must be passed to the Collector step explicitly from the preceding steps. If Collector is missing a particular file when processing a document page, that file cannot be copied to the document storage. In production, the PAL logs an error for a missing image file or an `XML` file and a warning for a missing `OCR` file.

To pass the required files to Collector, you need to assign Collector's input IA values *`InputImage`*, *`OcrDataCache`*, and *`InputXMLData`* with output of the preceding CaptureFlow steps. The necessary IA value assignments are configured when designing a CaptureFlow chart in CaptureFlow Designer. General instructions to assign IA values can be found in *OpenText Intelligent Capture - Designer Guide (ECPCORE-CPD)*.

The following example demonstrates how to perform all necessary IA value assignments for Collector to pass all required data to the PAL. The sample process has several steps, including:

- **Scan**: triggers the ScanPlus module
- **Classify**: triggers the Classification module
- **Recognition**: triggers the Extraction module
- **Validation**: triggers the Completion module
- **Collect**: triggers the Collector module
- **Export**: triggers Standard Export

The processing flow routes to Validation conditionally. All assignments are performed on level 0 (page level).

Table 6-5: Collector IA Value Assignments Required by *PAL*

Recognition step: IA value assignments	Validation step: IA value assignments
If data validation is needed, route to Validation:	*Route to Collector:*
• *`Recognition: 0.Image > Validation:0.Image`* • *`Recognition:0.OutputPageDataXml > Validation:0.PageDataXml`* • *`Classify:0.ImageOutputFileExtension > Collect:0.ImageOutputFileExtension`*	• *`Validation:0.Image > Collect:0.InputImage`* • *`Validation:0.PageDataXml > Collect:0.InputXMLData`* • *`Recognition:0.OcrDataCache > Collect:0.OCRDataCache`*

Recognition step: IA value assignments	Validation step: IA value assignments
If data validation is not needed, route to Export:	
• *Recognition:0.Image* > *Collect:0.InputImage* • *Recognition:0.OutputPageDataXml* > *Collect:0.InputXMLData* • *Recognition:0.OcrDataCache* > *Collect:0.OCRDataCache* • *Classify:0.ImageOutputFileExtension* > *Collect:0.ImageOutputFileExtension*	

Glossary

ANSI

American National Standards Institute

API

Application Programming Interface

DPI

Dots Per Inch

DPP

Dispatcher project file extension

HTML

HyperText Markup Language

IAP

InputAccel process file extension

IPP

Integrated ProcessFlow Project

JPEG

Joint Photographic Experts Group

MDF

Module Definition File

NL

New Line character

OCR-A

Optical Character Reader, Font A

OCR-B

Optical Character Reader, Font B

OCR

Optical Character Recognition

OMR

Optical Mark Recognition

PAL

Production Auto-Learning

PDA

Calera Processed Document Architecture file extension

PDF

Portable Document Format

TIFF

Tagged Image File Format

UNC

Universal Naming Convention

US

United States

UTC

Coordinated Universal Time

XML

Extensible Markup Language