STELLENT™

Creating Custom Conversion Engines

SDK-EN4-610

Table of Contents

CHAPTER 3: MODULE API SPECIFICATIONS

CHAPTER 4: GLOBAL VARIABLES

Chapter

1

OVERVIEW

INTRODUCTION

The information contained in this guide is based on Stellent™ Content Server 6.1. The information is subject to change as the product technology evolves and as hardware and operating systems are created and modified.

Due to the technical nature of browsers, web servers, and operating systems, Stellent, Inc. cannot warrant compatibility with all versions and features of third-party products.

This chapter contains these topics:

- About this Guide
- Stellent Product Distinctions
- If You Need Assistance

ABOUT THIS GUIDE

This guide provides information on creating custom conversion engines for the Refinery and lists the VB Module API specifications. It provides developers with the information they need to create and implement multiple custom conversion engines for the Refinery.

Audience

This guide is intended for application developers who need to create and implement custom conversion engines for the Refinery.

Conventions

The following conventions are used throughout this guide:

- The notation **<*install_dir*>/** is used throughout this guide to refer to the location on your system where Stellent Content Server product is installed.
- Forward slashes (/) are used to separate the directory levels in a path name. A forward slash will always appear after the end of a directory name.
- Notes, technical tips, important notices, and cautions use these conventions:

Symbol	Description
	This is a note. It brings special attention to information.
	This is a tech tip. It identifies information that can be used to make your tasks easier.

Symbol	Description
	This is an important notice. It identifies a required step or critical information.
	This is a caution. It identifies information that might cause loss of data or serious system problems.

STELLENT PRODUCT DISTINCTIONS

In this guide, the term *content server* is used generically to refer to both the Content Server and the Collaboration Server. The following table lists the distinctions of these two Stellent content management solutions:

Stellent Content Management Product and Feature Distinction

Product	Description
Stellent Content Server	A fully functional content management system providing end-to-end content management and personalized delivery of that content.
Stellent Collaboration Server	A fully functional content management system providing end-to-end content management and personalized delivery of that content. Additionally, a Stellent Collaboration Server license enables project-level security for collaborative authoring environments.

IF YOU NEED ASSISTANCE

The Stellent family of products is backed by a full range of support options to meet every business need. The service philosophy is to keep your Stellent environment fully operational by providing the best information and solutions available. The Stellent product support team consists of highly trained product engineers who excel at resolving complex technical issues. Every customer inquiry is tracked and managed through automated systems.

Important: The support options that are available for specific systems may vary, depending on the applicable service and maintenance agreements. Please refer to your contract for the support details for your Stellent system.

Support Options

You can choose from the following three support programs offered by Stellent:

- **Standard Maintenance and Support Program:** The standard support program is available during standard business hours domestically and internationally (Monday through Friday from 8 am to 5 pm for every time zone). It provides telephone and e-mail support for troubleshooting, bug fixes, call escalation, modifications, enhancements, and updates.
- **SDK Developer Support Program:** The SDK support program is available Monday through Friday from 8 am to 5 pm (Central Time in the USA, which is -6 hours from GMT). It provides telephone and e-mail support for customers who wish to use the Software Developer's Kit (SDK) to customize their Stellent systems.
- **Extended Support Program:** The extended support program provides the standard support services 24 hours a day and 7 days a week.

Note: Value Added Resellers (VARs) and Original Equipment Manufacturers (OEMs) may have different support programs in place.

Before Contacting Support

When you call or send e-mail, please provide the following information:

- Nature and severity of the problem.
- Stellent product and version.
- Serial number of the registered Stellent product.
- Operating system and version.
- Name and telephone number of the person the support engineers should contact if they need to call back.

In addition, depending on the situation, it may be helpful to know the following:

- Database type and version.
- Web browser type and version.
- Web server type and version.

Telephone

Technical support is available from the Support Hotline at 1-888-688-TECH (1-888-688-8324). The Support Hotline is accessible toll-free world-wide.

E-Mail

The Stellent support e-mail address is *support@stellent.com*. It is available for all technical support questions.

Internet

Technical support is also available through the Internet at http://support.stellent.com. You will be prompted for a username and password. To obtain a username and password, contact the Support Hotline at 1-888-688-TECH (1-888-688-8324).

Chapter

2

CUSTOM CONVERSION ENGINES

INTRODUCTION

This section provides a general description to assist developers in creating custom conversion engines in Visual Basic (VB) and other programming languages.

This chapter contains these topics:

- Input to the Conversion Engine
- Output from the Conversion Engine
- Sample Conversion Engine and Input/Output Files
- Constructing a Custom Conversion Engine in Visual Basic

INPUT TO THE CONVERSION ENGINE

The input to the custom conversion engine is an HDA file passed by the Refinery to the custom conversion engine. The `hasHDAInput` flag allows the

Refinery to pass an HDA file to the conversion engine via the command line. This action is performed by calling `loadGeneralSettings()` in the `Class_Initialize()` of your conversion class. For additional information, see “hasHDAInput” on page 2-13.

Tech Tip: When working with a programming language other than Visual Basic, the developer must load and store this data. Always perform data validation early in the conversion process to avoid data related errors later.

OUTPUT FROM THE CONVERSION ENGINE

The output from the custom conversion engine is an HDA file. The `hasHDAOutput` flag enables the Refinery to retrieve the results of a conversion in HDA format. For additional information, see “hasHDAOutput” on page 2-13.

The output has the following data:

```
StatusMsg=finished
OutExtension=<g_outFileExtension>
OutFormat=<g_outFormatType>
ResultFilePath=<g_currentOutFile>
LastErrorDescription=<g_lastErrorDescription>
ErrorStatusCode=<g_returnCode>
```

The `ErrorStatusCod`e instructs the Refinery how to handle the content item after the conversion. The conversion status is returned by the `g_returnCode` variable. For additional information, see “g_returnCode” on page 4-11.

All data must be present for the conversion to proceed successfully. If the conversion process fails, the output will set the following values:

```
StatusMsg=failed
```

```
OutExtension=
OutFormat=application/x-unknown
ResultFilePath=<Not Converted>
LastErrorDescription=<g_lastErrorDescription>
ErrorStatusCode=<g_returnCode>
```

This action is handled by calling `unloadClass()` in the `Class_Terminate()` of your conversion class. In other words, if `g_lastErrorDescription` has a value, the VB API will automatically create the correct output.

The `OutExtension` value is set to an empty string, the `g_lastErrorDescription` value is set to the error message to be displayed throughout the system, and the `ErrorStatusCode` returns the `g_returnCode` values. The custom conversion engine should load both `g_lastErrorDescription` and `g_returnCode`. For a list of return codes, see “Global Variables” on page 4-2.

Tech Tip: When working with a programming language other than Visual Basic, all paths will be passed in with forward slashes ‘/’ in the paths. When a content item is checked into an account, the @ symbol is represented in the HDA file as ‘\@.’ The custom conversion process will need to parse this carefully. For example: `D:/intradoc4/weblayout/groups/public/\@test/documents/adacct/000003~1.doc`

Outputting Multiple Files from the Conversion Engine

The conversion engine can output multiple files by defining the files as a semicolon separated list.

For example:

```
ResultFilePath=<file1>; <file2>; <file3>
```

However, only one file can be copied to the Weblayout directory as a web-viewable file.

The conversion engine can pick up the `ResultFilePath` by calling:

```
List = getStringSetting("ResultFilePath", " " )
```

Location of the Input and Output Files

The input HDA file path is passed as the only argument on the command line when the Refinery initiates the custom conversion. The output HDA file path is in the input HDA of the `outputhdaFile` setting. These files must remain in the specified locations.

Temporary Space Location

The temporary space location can be defined by calling `g_workingDirectory`.

Tech Tip: When working with a programming language other than Visual Basic, It is recommended that all temporary data be started in this directory:

```
<Refinery>\shared\ConversionEngines\<IDCName>
```

This directory path is represented in the HDA input file in the `dataExchangePath` setting. Using this directory ensures that files are in the correct Content Server when multiple Refineries are installed on a system.

Updating the Refinery Status Bar

The Refinery status bar can be updated by calling `updateStatus(msg As String)`.

Tech Tip: When working with a programming language other than Visual

Basic, note that the Refinery monitors the custom conversion process as well as the `outputhdaFile` data. During the conversion this file is used to update the status bar. The file should contain Properties with one value: `StatusMsg`. For example, the file may contain:

```
@Properties LocalData
StatusMsg=<Message to display>
@end
```

The Refinery evaluates changes in the file and updates the status bar when a change in the file occurs. The conversion process deletes and rewrites the file to update the status bar. For additional information, see “Constructing a Custom Conversion Engine in Visual Basic” on page 2-7.

Terminating a Subprocess

A subprocess can be terminated by calling `updateStatus(msg As String).`

Tech Tip: When working with a programming language other than Visual Basic, if a custom conversion process uses an ActiveX/OLE server as part of the conversion, the custom conversion process will capture the processID of the ActiveX/OLE server and write a Properties HDA to STDOUT.

The HDA data sent to STDOUT should contain the following values:

```
@Properties LocalData
Before=<space separated list of PIDs before the conversion starts>
After=<space separated list of PIDs after the conversion starts a sub process>
@end
```

- Ensure the *Before* value has a space separated list of ActiveX/OLE server ProcessIDs before the conversion starts the ActiveX/OLE server. Often this list is empty.
- Ensure the *After* value has a space separated list of ActiveX/OLE server ProcessIDs after the conversion starts the ActiveX/OLE server. This list often contains one PID.

The Refinery will evaluate the difference between the two lists and terminates the correct ProcessID if the custom conversion process times out.

SAMPLE CONVERSION ENGINE AND INPUT/ OUTPUT FILES

Conversion Engine

The sample Visual Basic conversion engine is provided on the Refinery CD in the \samples\Sample_Conversion_Engine\ directory.

Input HDA File

A sample input file named DocConverter.hda is provided on the Refinery CD in the \samples\Sample_Conversion_Engine\native\sampleInput\ directory.

For additional information, see “hasHDAInput” on page 2-13.

The name/value pairs are loaded into the `g_documentDataColl` collection after calling `initConverter()` and can be accessed using these function:

- getStringSetting()
- getBoolSetting()
- getIntSetting()

- ❖ getValueFromParameters()

Output HDA File

A sample output file named DocConverterResults.hda is provided on the Refinery CD in the \samples\Sample_Conversion_Engine\native\sampleOutput\ directory.

The data in that file is set by the contents of the listed variables. The output HDA file is written by calling `unloadClass()` in the Private Sub `Class_Terminate()` of the conversion class. For additional information, see “hasHDAOutput” on page 2-13.

CONSTRUCTING A CUSTOM CONVERSION ENGINE IN VISUAL BASIC

Follow these steps to construct a custom conversion engine in Visual Basic for the Refinery:

1. Create an entry in the DocumentConversions table of the custom component.
2. Create an entry in the ConversionSteps table of the custom component.
3. Create a ConvertMain module containing a Public Sub Main().
4. Create a ConverterCls class containing the custom conversion code.
5. Define the functions in the three provided modules.

Steps one and two should be performed using the Component Wizard.

The Component Wizard is a stand-alone application and is executed locally from the server.

- To launch the Component Wizard from NT select *Start—Programs—Stellent Content Server—Master_on_<server>—Utilities—Component Wizard.*
- To launch the Component Wizard from Solaris navigate to the <*home*>/bin directory. At the prompt enter:
  `IntradocMS_ComponentWizard`

Note: Click **Help** to view the online guide designed specifically for the Component Wizard.

Step One: Create an Entry in the DocumentConversions Table

The developer must define the new conversion engine in a custom component. Create an entry in the DocumentConversions table of the custom component. The default location for this tables is `<home>\config\resources\std_docrefinery.htm`. This entry has four parts, for example:

```
@ResultSet DocumentConversions
4
drConversion
drSteps
drDescription
drIsEnabledFlag
SampleConversion
SampleConversionStep, PostscriptToPDF
Example Custom Conversion Engine
TRUE
@end
```

drConversion

The `drConversion` entry is the name of the conversion. This is also displayed in the Configuration Manager under the File Formats tab.

drSteps

The `drSteps` entry is a comma-separated list of steps. This is defined in the ConversionSteps table and must be run to complete the conversion. This entry is IdocScript enabled.

For example, if the entry reads:

```
<$if UseAdobePDFMaker$>WordToPDF
<$else$>MSOfficeToPostscript, PostscriptToPDF
<$endif$>, CreatePDFThumbnail
```

- If `UseAdobePDFMaker` is TRUE in the *intradoc.cfg* file then the system executes the `WordToPDF` step.
- If `UseAdobePDFMaker` is FALSE in *the intradoc.cfg* file then the system executes the `MSOfficeToPostscript` and `PostscriptToPDF` steps.

drDescription

The `drDescription` entry is a description of the conversion.

drIsEnabledFlag

The `drIsEnabledFlag` entry should be set to TRUE.

Step Two: Create an Entry in the ConversionSteps Table

Create an entry in the ConversionSteps table with a custom component. The developer must define nine entries, for example:

```
@ResultSet ConversionSteps
9
drStep
drStepType
drStepAction
drStepParameters
drStepTimeoutName
drStepReport
drStepControlCodes
drStepDescription
drStepIsEnabled
SampleConversionStep
commandLine
SampleEngine.exe
-engine <$drConversion$> -key1 val1
DefaultNativeTimeout
Converting <$drConversion$>
hasHDAInput, hasHDAOutput, onErrorFail
This is a Sample Step
TRUE
@end
```

drStep

The `drStep` entry is the name of the step referenced in the DocumentConversions table as the `drStep` entry.

drStepType

The `drStepType` entry defines the type of step. Set this entry to `commandLine`.

drStepAction

The `drStepAction` entry is the name of the custom engine executable file. The executable file should be placed in the directory indicated by the DocConverterEngineDir setting of the *intradoc.cfg* file. The Refinery will expect to find the executable file in that directory.

Important: If a full path is not defined, the executable should be the DocConverterEngineDir.

drStepParameters

The `drStepParameters` entry is a string passed to the conversion engine via the input HDA file. This entry is IdocScript enabled and is evaluated by the Refinery before passing the information to the conversion engine. Values within the string should be defined using double-quotation marks. For example, `-key1 "val1" –key2 "val2" –key3 "val3"`. See “drStepControlCodes” on page 2-12, for possible values.

drStepTimeoutName

The `drStepTimeoutName` entry is the name of the timeout factor in the *intradoc.cfg* file. As files are processed by the Refinery, the file is allotted time based on the size of the file and these settings. The timeout factor in minutes is constructed according to the size of the file in MB times 200 multiplied by the timeout factor (FILESIZE MB * 200 * FACTOR). The file is allotted at least the minutes indicated by the 'min' column, but no more minutes than indicated by the 'max' column.

This setting indicates which set of timeouts will be used. A timeout set is defined in the *idcrefinery.cfg* file by specifying the timeout name and the minimum, maximum, and factor entry. For example, if this value is set to "MyTimeouts" then the *idcrefinery.cfg* file entry would be set to:

```
MyTimeouts#min=60 MyTimeouts#max=60 MyTimeouts#factor=3
```

drStepReport

The `drStepReport` entry is the initial status line on the bottom of the Refinery. This entry is IdocScript enabled.

drStepControlCodes

The `drStepControlCodes` entry is a list of comma-separated flags that provide the Refinery information concerning the step. These flags should be included in the `drStepParameters` definition: `onErrorFail`, `hasHDAInput` (input to the conversion engine), and `hasHDAOutput` (output to the conversion engine).

onErrorFail

The `onErrorFail` flag instructs the Refinery to end the current task when the conversion process fails.

hasHDAInput

The `hasHDAInput` flag allows the Refinery to pass an HDA file to the conversion engine via the command line.

The HDA file is made up of the following data:

- All name/value pairs for the Refinery of the *intradoc.cfg* and the *idcrefinery.cfg* files
- IdocScript evaluated `drStepParameters` from the ConversionSteps table
- Path native file
- Potential path to the web vault. The extension will be the native file extension
- Name of the current conversion
- File name and title

hasHDAOutput

The `hasHDAOutput` flag enables the Refinery to retrieve the results of a conversion in HDA format. For additional information, see the sample Output File on page 1-2.

The output has the following data:

```
StatusMsg=finished
OutExtension=<g_outFileExtension>
OutFormat=<g_outFormatType>
ResultFilePath=<g_currentOutFile>
LastErrorDescription=<g_lastErrorDescription>
ErrorStatusCode=<g_returnCode>
```

The `ErrorStatusCode` instructs the Refinery how to treat the file after the conversion. The conversion status is returned by the `g_returnCode` variable. For additional information, see the “Global Variables” on page 4-2.

The `g_returnCode` can have one of the following values:

Name	Value
CONVERSION_GEN_SUCCESS	Indicates the conversion engine was successful.
CONVERSION_NO_CODE	Used for backward compatibility.
CONVERSION_GEN_FAILURE	Indicates the conversion failed and the Refinery should handle the failure as the step definition was defined.
CONVERSION_FORCE_PASSTHRU	Instructs the Refinery to perform a passthru on the file. The `FORCE` values override the step definitions.
CONVERSION_FORCE_FAILURE	Instructs the Refinery to fail the file. The `FORCE` values override the step definitions.
CONVERSION_ERROR_CONTINUE	Instructs the Refinery to continue to the next conversion step even though this step has one or more errors.

drStepDescription

The `drStepDescription` entry provides a description of the step.

drStepIsEnabled

The `drStepIsEnable` entry enables the step and should be set to TRUE.

Step Three: Create a ConvertMain Module

Create a standard VB project with a ConvertMain module containing a Public Sub `Main()` that starts the conversion engine. These components must be provided by the developer and have strict requirements that you must follow.

The Sub `main()` of the converter engine must do the following:

1. Declare the converter class:
   `Dim SamplConverter As SamplConverterCls`
2. Initialize converter variables and load the HDA input file:
   `initConverter <pathTohdaInputFile>`
3. Initialize the custom class:
   `Set SamplConverter = New SamplConverterCls`
4. Call the `convert()` member of the converter class:
   `SamplConverter.convert`
5. Close the converter class:
   `Set SamplConverter = Nothing`

Step Four: Create a ConverterCls Class

Create a custom `ConverterCls` class that will contain the custom conversion code. These components must be provided by the developer and have strict requirements that you must follow.

Class_Initialize()

The Private Sub `Class_Initialize()` of the custom converter class must perform these listed actions:

- Load the general conversion settings using:
  `loadGeneralSettings`

- If needed, load any specialized settings required for the conversion using: `loadPrivateSettings`
- Update the Refinery status bar using: `updateStatus "Sample Converter Loaded"`

Class_Terminate()

The Private Sub `Class_Terminate()` of the custom converter class must perform this action:

- Unload the class and write output file using: `unloadClass`

convert()

The Public Sub `convert()` of the custom converter class must perform this action:

- Begin the conversion process. The developer specifies the requirements based on system needs.

Step Five: Define the Functions in the Provided Modules

A Visual Basic conversion engine consists of several provided modules. For additional information, see “Module API Specifications” on page 3-1.

- The developer must define the functions in these provided modules.
- The developer must add the modules in `<Refinery_cdrom>/samples/Sample_Conversion_Engine/native/VBUtilities to their project.`

CommonUtils

This module contains miscellaneous functions needed by a conversion engine.

ComunicationUtils

This module contains functions used to get data from the HDA input file.

PowerPrn_Mod

This module contains functions used to configure the printer.

RegistryUtils

This module contains functions used to get and set registry settings.

ReuseApps

This module contains functions used to termination a process.

Chapter

3

MODULE API SPECIFICATIONS

INTRODUCTION

These application interface specifications consist of modules that contain functions needed by the conversion engine, used to retrieve data from the input file, to configure the printer., to get and set registry settings, and to termination a process.

This chapter contains these topics:

- CommonUtils Module
- ComunicationUtils Module
- PowerPrn_Mod Module
- RegistryUtils Module
- ReuseApps Module

COMMONUTILS MODULE

This module contains miscellaneous functions needed by a conversion engine.

appSnapShot

Description

Captures the ProcessID of the ActiveX/OLE Server that is used by the conversion engine to process the file.

- The entry `coll` is an empty but initialized VB Collection.
- After `appSnapShot` is called, the passed-in collection is assigned a list of all ProcessIDs whose name matches the value of `g_convExeName`.
- This is normally called before and after starting an ActiveX/OLE Server.

Follow these steps to initialize:

1. Set `g_convExeName` to the name of the process.

Note: The first call `appSnapShot g_PIDBeforeCollection` collects the currently running ProcessIDs with the name in `g_convExeName`.

2. Start the native application the conversion process needs to use.
3. Call `appSnapShot g_PIDAfterCollection` to gather the ProcessID of the ActiveX/OLE Server the conversion engine is using.
4. Call `updateInUsePIDs` to send the information needed by the Refinery to terminate the ActiveX/OLE Server used by the conversion engine should the conversion timeout.

Definition

Function As Boolean

Name	Type
`Coll (ByRef)`	Collection

Example

```
g_convExeName = “WORD.EXE”
appSnapShot g_PIDBeforeCollection
Set converterObj = CreateObject("word.application")
appSnapShot g_PIDAfterCollection
updateInUsePIDs
```

checkForExclusiveAccess

Description

Determines whether a file is in use by another process.

- The `filePath` entry should be a properly formatted path to an existing file.
- The function returns TRUE if the file can be opened for exclusive access.
- The function returns FALSE if the file does not have exclusive access.

Definition

Function As Boolean

Name	Type
`filePath` (ByVal)	String

checkPath

Description

Determines what kind of path is in fName.

- The fName entry is a path to a file.
- The function will return one of the following constants:

Name	Type	Value
`PATH_NOT_EXIST`	Global Constant as Integer	0
`PATH_IS_DIRECTORY`	Global Constant as Integer	1
`PATH_IS_READ_ONLY_FILE`	Global Constant as Integer	2
`PATH_IS_WRITABLE_FILE`	Global Constant as Integer	3

Definition

Public Function `checkPath` returns a String

Parameter Name	Parameter Type
`checkPath(fName)`	String

clearLog()

Description

Clears the log.

- Deletes the temporary log written by the conversion engine.

Definition

Public Sub

convertToJavaStylePath

Description

Converts path to a Java style pathname.

- Searches and replaces the backslash character in the pathname with a forward slash (/).

Definition

Public Function As String

Name	Type
`path (ByVal)`	String

convertToOldStylePath

Description

Converts path to a DOS style pathname.

- Replaces the forward slash character in the pathname with a backslash (\ ).

Definition

Public Function As String

Name	Type
`path (ByVal)`	String

createTempSaveAsName()

Description

Returns a temporary path name.

- A temporary path name is created using the Refinery working path and the current content item revision ID (dID).
- This temporary path name can be used by native applications to save the content item. This allows the content item to be easily identified in the print queue.

Definition

Public Function returns a string.

deleteFileIgnoreError

Description

Deletes the file in `fName`.

- The entries in the string `fName` are deleted and the error ignored.

Definition

Public Sub

Name	Type
`fName (ByVal)`	String

execCmdNoWait

Description

Launches `cmdline` without waiting for execution.

- The Refinery is automatically updated that the new process has executed.
- The default for `hide` is FALSE.
- If the process fails to launch -1 is returned.
- The function returns the Process ID of the newly launched program.
- If `hide` is TRUE, the newly launched program runs hidden.
- If `m_debugMode` is TRUE, `hide` is set to FALSE.

Definition

Public Function (Long)

Name	Type
`cmdline`	String
`hide (optional)`	Boolean
`m_debugMode`	Boolean

execCmdWait

Description

Launches `cmdline` and waits the number milliseconds defined in `waitTime`.

- The Refinery is automatically updated that the new process has executed.
- The default for `hide` is FALSE.
- If the process fails to launch -1 is returned.
- The function returns TRUE, if `cmdline` finishes in `waitTime`.
- If `hide` is TRUE, the newly launched program runs hidden.
- If `m_debugMode` is TRUE, `hide` is set to FALSE.
- The Process ID of the newly launched program is automatically sent to STDOUT.

Definition

Public Function As Boolean

Name	Type
`cmdline`	String
`waitTime`	Long
`hide (optional)`	Boolean
`m_debugMode`	Boolean

execCmdWaitDDE

Description

Launches `cmdline`, waits the number milliseconds defined in `waitTime`, and passes DDE message through.

- The Refinery is automatically updated that the new process has executed.
- The default for `hide` is FALSE.
- If the process fails to launch -1 is returned.
- The function returns TRUE if `cmdline` finishes in `waitTime`.
- If `hide` is TRUE, the newly launched program runs hidden.
- If `m_debugMode` is TRUE, `hide` is set to FALSE.
- The Process ID of the newly launched program is automatically sent to STDOUT.

Note: The most versatile execCmd* function is `execCmdWaitDDE`. Use this function if you are unsure which to use.

Definition

Public Function As Boolean

Name	Type
`cmdline`	String
`waitTime`	Long
`hide`	Boolean
`m_debugMode`	Boolean

fileCopyWithRetry

Description

Copies the string information contained in `fromPath` to `toPath`.

- If the copy is successful TRUE is returned.
- If the copy is not successful FALSE is returned.
- The copy command will be tried 150 times before it terminates.

Definition

Public Function As Boolean

Name	Type
`fromPath`	String
`toPath`	String

findRunningPID()

Description

Finds the ProcessID of the ActiveX/OLE server in use by the conversion.

- Finds the ProcessID of the newest ActiveX/OLE server by finding the difference between `g_PIDBeforeCollection` and `g_PIDAfterCollection`.
- The conversion can be used to terminate the process.

Definition

Public Function as Integer

initConverter

Description

Initializes a converter.

- The input HDA`File` is passed from the Refinery to the VB Converter via the command line.
- Call from the Sub `main()` of the VB Converter.

Definition

Public Sub

Name	Type
`hdaFile`	String

isAppRunning

Description

Queries whether the executable defined in `exeName` is running.

- The function returns the ProcessID if the executable is running.
- The function returns -1 if the executable is not running.

Definition

Public Function as Long

Name	Type
`ExeName`	String

isFileAccessible

Description

Calls `checkForExclusiveAccess()` to determine whether a file is in use by another process.

- Checks the path for up to one second. If the path becomes available during that one second space, it returns TRUE, otherwise it returns FALSE.
- The `path` entry should be a properly formatted path to an existing file.
- The function returns TRUE if the path is not locked.
- The function returns FALSE if the path is locked.

Definition

Public Function As Boolean

Name	Type
`path`	String

loadLocalhdaFile

Description

Loads the local properties of an HDA file into a `g_documentDataColl` collection.

- The function returns TRUE if the load was successful.
- The function returns FALSE if the load was not successful.
- Called by `initConverter (hdaFile As String)`.

Definition

Public Function As Boolean

Name	Type
hdaFile	String

makeCleanNTPath

Description

Creates a clean NT file pathname.

- Removes the following illegal NT pathname characters from the path specification:

Character	Description
/	Forward slash
:	Colon (see note)
*	Asterisk
"	Quotes
<	Less than
>	Greater than
|	Bar
.	Period
	Blank space

Note: If the second character is a colon the path remains valid.

Definition

Public Function As String

Name	Type
path	String

parseToArray

Description

Parses `str` into an array.

- Members of the array must be separated by `tok`.
- The first element of `strarray` is 0. In releases prior to 4.5, strarray(0) was left empty and strarray(1) contained the first element.
- The last element of the array cannot be an empty string. If `tok` is the last character of `str`, a blank element will NOT be added to the strarray().

Definition

Public Sub

Name	Type
str	String
tok	String
I (ByRef)	Integer
strarray() (ByRef)	String

sendPIDsToSTDOUT

Description

Sends the ProcessIDs to STDOUT.

- Writes the ProcessIDs of the ActiveX/OLE Server to STDOUT for possible termination by the Refinery.
- The option xPID can be used if another launched process needs to be terminated by the Refinery.

Definition

Public Sub

Name	Type
`xPID (optional)`	String

sendToSTDOUT

Description

Sends the information to standard output.

- Writes the string `sOut` to STDOUT.

Definition

Public Sub

Name	Type
`sOut (ByVal)`	String

splitFileNameDir

Description

Parses out the `base` file name from `fName`.

- The string `fName` is a full complete path to a file and is left unchanged.
- The string `base` is passed in empty and is set to `fName` without the complete path or extension.
- The function returns the full path to `fName`.

Definition

Function As String

Name	Type
`fName`	String
`base (ByRef)`	String

splitFileNameExt

Description

Parses the filename and extension from `fName`.

- The string `fName` is a full complete path to a file and is unchanged.
- The function returns `fName` without the extension.
- The string `ext` is passed in empty and is set to the extension of fName.

Definition

Function As String

Name	Type
`fName`	String
`ext (ByRef)`	String

storeSelfInUsePID()

Description

Finds the ProcessID of your currently running custom converter and adds the ProcessID to the list of processes the converter has started.

- Automatically called by `loadGeneralSettings()`.

Definition

Public Sub

unloadClass()

Description

Handles the termination of the conversion engine.

- Should be called by the Private Sub `Class_Terminate()` of the conversion engine. It performs these functions to handle the termination of the conversion engine:
- Reassigns the original default system printer if the printer setting was changed.
- Deletes the temp file in `g_tempSaveAs`.

- Writes the output HDA file for the Refinery.

Note: The Private Sub `Class_Terminate()` of your custom conversion class must call `unloadClass()` for this to be executed.

Definition

Public Sub

updateInUsePIDs

Description

Adds the ProcessID of the ActiveX process or a parameter to the "*Process ID in Use"* list.

- When called without a parameter, the ProcessID of the ActiveX current process is added to the ProcessID in use list. This process is named by `g_convExeName`.
- When called with a parameter, the parameter is added to the list.
- Dynamically updates STDOUT entry. For example, in Sample Conversion Engine after starting an ActiveX server, `updateInUsePIDs` should be called rather than `sendPIDsToSTDOUT()`
- The option `xPID` can be used if another process is launched that may need to be terminated by the Refinery.

Definition

Public Sub

Name	Type
`xPID (optional)`	Long

writeLogEntry

Description

Creates a log entry.

- Enters `msg` as in the temporary converter engine log.
- The entry `msgType` is optional and can be ignored.

Definition

Public Sub

Name	Type
`msg`	String
`msgType (optional)`	String

COMUNICATIONUTILS MODULE

This module contains functions used to get data from the HDA input file.

boolToStrInt

Description

Returns the Boolean value as a string.

- TRUE returns 1.
- FALSE returns 0.

Definition

Public Function As String

Name	Type
b	Boolean

cleanErrorStr

Description

Removes the string `removeTok` from `g_lastErrorDescription`.

- Used to remove line control characters such as line-feed Chr(10) or new-line Chr(13) from the error string returned by the ActiveX/OLE server.
- Called by Public Sub `writeResults()`.

Definition

Private Sub

Name	Type
removeTok	String

getBoolSetting

Description

Returns the Boolean value from the input HDA file.

- If `name` is not set, the value for `default` is returned.

Definition

Public Function As Boolean

Name	Type
`name`	String
`default`	Boolean

getIntSetting

Description

Returns the integer value from the input HDA file.

- If `name` is not set, the value for `default` is returned.

Definition

Public Function as Integer

Name	Type
`name`	String
`default`	Integer

getLine

Description

Reads and returns a line from a file opened by `fn`.

- The line must be terminated using the line control characters line-feed Chr(10) or new-line Chr(13).

Definition

Public Function As String

Name	Type
fn	Integer

getStringSetting

Description

Returns the string value from the input HDA file.

- If `name` is not set, the value for `default` is returned.

Definition

Public Function As String

Name	Type
name	String
default	String

getSystemDir

Description

Provides system directory path.

- The function returns the path to the \system32 directory.

Definition

Public `Function getSystemDir` returns a String

getValueFromParameters

Description

Returns the value on the pseudo commandline `g_stepParameters`.

- If key is not on the line, an empty string is returned.

Definition

Public Function As String

Name	Type
`key`	String

Example

Returns the value on the pseudo commandline and sets `testval` to *value1*:

```
g_stepParameters = -key1 value1 –key2 value2
Dim testval as String
testval = getValueFromParameters(“key1”)
```

loadGeneralSettings

Description

Loads settings needed by all conversion engines.

- Calling `loadGeneralSettings()` will initialize a number of variables including:

  `g_inFile`g_currentOutFile

  `g_convType`g_verboseMode

  `g_dataExchangePath`g_debugMode

`g_printerPortPath`g_printerPortPath
`g_workingDirectory`

Note: The Private Sub `Class_Initialize()`of your custom conversion class must call `loadGeneralSettings()` to initialize these variable.

Definition

Public Sub

writeResults

Description

Writes the results of the custom conversion engine.

- Called by `Public Sub unloadClass()`.
- Creates an output HDA file with the results of the query.

Note: The Private Sub `Class_Terminate()`of your custom conversion class must call `unloadClass()` to complete the custom conversion.

Definition

Public Sub

POWERPRN_MOD MODULE

This module contains functions used to configure the printer.

isValidPrinter()

Description

Writes the results of the custom conversion engine.

- Searches whether a valid printer is installed on the system. Called by the setSystemDefPrinter() method.
- Returns TRUE if the prnName is installed
- Returns FALSE if the prnName is not installed.

Definition

Public Function As Boolean

Name	Type
`prnName`	String

setSystemDefPrinter()

Description

Defines the printer settings by specifying the printer and the mode. The mode parameter must be one of these two constants:

Global Const SET_PRINTER_START_CONVERT As Integer = 1

Global Const SET_PRINTER_FINISH_CONVERT As Integer = 2

- If SET_PRINTER_START_CONVERT is passed, the call does extra error checking before setting the printer, it calls isValidPrinter() to ensure the printer is actually installed on the system, if the printer is not installed -1 is returned. It also ensures the port the printer will print to matches the Refinery PrinterPortPath setting, if they do not match -2 is returned. Otherwise 0 is returned.
- If PRINTER_FINISH_CONVERT is passed, then the printer is switched with no error checking. and it always returns 0.

Definition

Public Function as Integer

Name	Type
`sPrinter`	String
`mode`	Integer

setSystemPrinterToRefine()

Description

Sets the system printer.

- Serves as the way to set the system printer to be used by the conversion engines. Called by the convert() method of the conversion class.
- Return TRUE if the system printer could be set.
- Returns FALSE if unable to set the printer.

Definition

Public Function As Boolean

REGISTRYUTILS MODULE

This module contains functions used to get and set registry settings.

In these functions `lPredefinedKey` should be one of the following main registry keys:

- KEY_CLASSES_ROOT
- HKEY_LOCAL_MACHINE
- HKEY_CURRENT_USER
- HKEY_USERS

CreateNewKey

Description

Creates a new registry key using the string `sNewKeyName`.

Definition

Public Function

Name	Type
`lPredefinedKey`	Long
`sNewKeyName`	String

Example

```
CreateNewKey HKEY_LOCAL_MACHINE, "TestKey\SubKey1\SubKey2"
```

DeleteValue

Description

Deletes the registry value of the string `sValueName`.

Definition

Public Sub

Name	Type
`lPredefinedKey`	Long
`sKeyName`	String
`sValueName`	String

Example

```
DeleteValue HKEY_LOCAL_MACHINE,TestKey\SubKey1\SubKey2",
"KeyName"
```

DeleteKey

Description

Deletes the registry key under `sKeyName`.

Definition

Public Sub

Name	Type
`lPredefinedKey`	Long
`sKeyName`	String

Example

```
DeleteKey HKEY_LOCAL_MACHINE, "TestKey\SubKey1\SubKey2"
```

QueryValue

Description

Queries the registry key value.

- The function returns the contents of the `sValueName` under the registry key `sKeyName`.

Definition

Public Function

Name	Type
`lPredefinedKey`	Long
`sKeyName`	String
`sValueName`	String

RegKeyExist

Description

Checks for registry sub-key.

- The function returns TRUE if `sKeyTarget` is a sub-key of `sKeyName`.
- The function returns FALSE if `sKeyTarget` is not a sub-key.

Definition

Public Function As Boolean

Name	Type
lPredefinedKey	Long
sKeyName (ByVal)	String
sKeyTarget (ByVal)	String

SetKeyValue

Description

Sets registry key value.

- Sets the registry value sValueName under sKeyName to be vValueSetting.
- Sets the type of lValueName to either REG_SZ or REG_DWORD.

Definition

Public Function

Name	Type
lPredefinedKey	Long
KeyName	String
sValueName	String
vValueSetting	Variant
ValueType	Long

Example

```
SetKeyValue HKEY_LOCAL_MACHINE, "TestKey\SubKey1",
"StringValue", "Hello", REG_SZ
```

REUSEAPPS MODULE

This module contains functions used to termination a process.

terminateCurrentActiveXProc()

Description

Terminates the current ActiveX server started by your custom converter.

- Requires that `g_PIDBeforeCollection` and `g_PIDAfterCollection` have been set.

Definition

Public Sub

Name	Type
`procID`	Long

terminateProcByPID()

Description

Terminates a process.

- The process is terminated by force using `procID`.
- Prior to release 4.5, `killPID()` was used to terminate a process.

Definition

Public Sub

Name	Type
procID	Long

Chapter

4

GLOBAL VARIABLES

INTRODUCTION

This chapter provides the available global variables. Each variable is described and defined by type, whether it is editable, and whether it is auto loaded.

These are the global variables described in this chapter:

g_additionalData	g_inFile	g_printerDriver
g_baseConnectionDir	g_lastErrorDescription	g_printerPortPath
g_convType	g_outFileExtension	g_returnCode
g_currentOutFile	g_outFormatType	g_stepParameters
g_dataExchangePath	g_PIDAfterCollection	g_verboseMode
g_debugMode	g_PIDBeforeCollection	g_workingDirectory

GLOBAL VARIABLES

g_additionalData

Description

Allows steps to pass data ahead to future steps in the refining process.

Includes an additional entry in the step results (DocConverterResults.hda). The additional data entry allows steps to pass data ahead to future steps in the refining process. This entry can be empty (null), but it must be present in the step results.

- Uses the drStepControlCode flag *onErrorAttemptSecondaryConversion*.
- If a conversion fails with the *onErrorAttemptSecondaryConversion* flag set, an alternate step can be executed.
- An alternate step can be assigned by setting the *SecondaryConversionStepName* entry. This entry is assigned a value of "OIXConverter" by default.

Definition

Variable As String

Handling	Description
Editable	Yes
Auto Loaded	No

g_baseConnectionDir

Description

Provides the directory path to the connection. The directory path to the connection is defined by referencing the *intradoc.cfg* file.

Definition

Variable As String

Handling	Description
Editable	No
Auto Loaded	Yes

g_convExeName

Description

Provides the executable name of the ActiveX/OLE server. The executable name of the ActiveX/OLE server used in the conversion is the process name.

Definition

Variable As String

Handling	Description
Editable	Yes
Auto Loaded	No

g_convType

Description

Provides the conversion type.

- ❖ Use this to verify that the correct process is running.
- ❖ Can be used to perform several types of conversions in an executable.

Definition

Variable As String

Handling	Description
Editable	No
Auto Loaded	Yes

g_currentOutFile

Description

Provides the output file path. The output HDA file path is in the input HDA of the `outputhdaFile` setting. These files must remain in the specified locations.

Definition

Variable As String

Handling	Description
Editable	Yes
Auto Loaded	No

g_dataExchangePath

Description

Provides the data exchange path (the directory where the VB Component and the Refinery can exchange data).

Definition

Variable As String

Handling	Description
Editable	No
Auto Loaded	Yes

g_debugMode

Description

Provides debug mode state.

- This relates to the Indexer Debug Level. The more debug information listed in the server window, the slower the indexing progresses.
- The function returns TRUE if debug mode is on.
- The function returns FALSE if debug mode is off.

❖ This list shows the debug levels from the least to the most debug information:

Indexer Debug Level	Description
none	No information for each file access is displayed, and no log will be generated.
verbose	Displays information for each file accessed. Indicates indexed, ignored, or failed, and generates a full report.
debug	Displays the medium level of information, which is specifically functional.
trace	Displays the lowest level of information for each activity performed.
all	Displays the highest level of debug information.

Definition

Variable As Boolean

Handling	Description
Editable	No
Auto Loaded	Yes

g_inFile

Description

Provides the input file path (the directory path to the native vault).

Definition

Variable As String

Handling	Description
Editable	No
Auto Loaded	Yes

g_lastErrorDescription

Description

Provides a description of the most recent error. This text will also be used as the status error in the Refinery and related logs.

Definition

Variable As String

Handling	Description
Editable	Yes
Auto Loaded	No

g_outFileExtension

Description

The file extension of the converted or web-viewable file.

Definition

Variable As String

Handling	Description
Editable	Yes
Auto Loaded	No

g_outFormatType

Description

The format type of the converted or web-viewable file.

Definition

Variable As String

Handling	Description
Editable	Yes
Auto Loaded	No

g_PIDAfterCollection

Description

Provides a list of ProcessIDs after initialization.

The list of ProcessIDs for executables that were running after the ActiveX/OLE server was initialized. For additional information, see “g_convExeName” on page 4-3.

Note: `g_PIDBeforeCollection` - `g_PIDAfterCollection` = the

ProcessID of the ActiveX/OLE server used by the conversion engine.

Definition

Collection

Handling	Description
Editable	Yes
Auto Loaded	No

g_PIDBeforeCollection

Description

Provides a list of ProcessIDs prior to initialization.

The list of ProcessIDs for executables that were running before the ActiveX/OLE server was initialized. For additional information, see “g_convExeName” on page 4-3.

Note: `g_PIDBeforeCollection` - `g_PIDAfterCollection` = the ProcessID of the ActiveX/OLE server used by the conversion engine.

Definition

Collection

Handling	Description
Editable	Yes
Auto Loaded	No

g_printerDriver

Description

The name of a system printer you wish to use.

Definition

Variable As String

Handling	Description
Editable	Yes
Auto Loaded	No

g_printerPortPath

Description

The path to the Distiller printer port.

Definition

Variable As String

Handling	Description
Editable	No
Auto Loaded	Yes

g_returnCode

Description

Provides return codes. Instructs the Refinery how to handle the converted file.

Definition

Variable As Integer

Handling	Description
Editable	Yes
Auto Loaded	No

Returns one of the following values:

Name	Type	Editable	Auto Loaded	Value
CONVERSION_GEN_SUCCESS	Integer	No	Yes	1
CONVERSION_NO_CODE	Integer	No	Yes	0
CONVERSION_GEN_FAILURE	Integer	No	Yes	-1
CONVERSION_FORCE_PASSTHRU	Integer	No	Yes	-2
CONVERSION_FORCE_FAILURE	Integer	No	Yes	-3
CONVERSION_ERROR_CONTINUE	Integer	No	Yes	-4

g_stepParameters

Description

Provides a list of “command line” parameter passed in to the conversion engine.

- This format is used:
  `-key1 “value1” -key2 “value2” –key3 “value3”`
- The line is defined in the definition of the current step and is IdocScript enabled.
- Use `getValueFromParameters()` to read values off this line.

Definition

Variable As String

Handling	Description
Editable	No
Auto Loaded	Yes

g_verboseMode

Description

Provides the verbose mode state.

- This relates to the Indexer Debug Level. The more debug information listed in the server window, the slower the indexing progresses.
- The function returns TRUE if verbose logging is ON.
- The function returns FALSE if verbose logging is OFF.

❖ This list shows the debug levels from the least to the most debug information:

Indexer Debug Level	Description
none	No information for each file access is displayed, and no log will be generated.
verbose	Displays information for each file accessed. Indicates indexed, ignored, or failed, and generates a full report.
debug	Displays the medium level of information, which is specifically functional.
trace	Displays the lowest level of information for each activity performed.
all	Displays the highest level of debug information.

Definition

Variable As Boolean

Handling	Description
Editable	No
Auto Loaded	Yes

g_workingDirectory

Description

Provides the working temporary directory.

Definition

Variable As String

Handling	Description
Editable	No
Auto Loaded	Yes

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X