Kofax Customer Communications Manager

Core Scripting Language Developer's Guide

Version: 5.2

Date: 2018-11-23

Table of Contents

Preface

This guide provides detailed information about the Kofax Customer Communications Manager Core scripting language.

Related documentation

The documentation set for Customer Communications Manager is available here:[1]

https://docshield.kofax.com/Portal/Products/CCM/520-nz7r6s9geq/CCM.htm

In addition to this guide, the documentation set includes the following items:

Kofax Customer Communications Manager Release Notes
Contains late-breaking details and other information that is not available in your other Kofax Customer Communications Manager documentation.

Kofax Customer Communications Manager Core Developer's Guide
Provides a general overview and integration information for CCM Core.

Kofax Customer Communications Manager Getting Started Guide
Describes how to use Contract Manager to manage instances of Kofax Customer Communications Manager.

Help for Kofax Customer Communications Manager Designer
Contains general information and instructions on using Kofax Customer Communications Manager Designer, which is an authoring tool and content management system for Kofax Customer Communications Manager.

Kofax Customer Communications Manager Repository Administrator's Guide
Describes administrative and management tasks in Kofax Customer Communications Manager Repository and Kofax Customer Communications Manager Designer for Windows.

Kofax Customer Communications Manager Repository User's Guide
Includes user instructions for Kofax Customer Communications Manager Repository and Kofax Customer Communications Manager Designer for Windows.

[1] You must be connected to the Internet to access the full documentation set online. For access without an Internet connection, see "Offline documentation" in the Installation Guide.

Kofax Customer Communications Manager Repository Developer's Guide
Describes various features and APIs to integrate with Kofax Customer Communications Manager Repository and Kofax Customer Communications Manager Designer for Windows.

Kofax Customer Communications Manager Template Scripting Language Developer's Guide
Describes the CCM Template Script used in Master Templates.

Kofax Customer Communications Manager API Guide
Describes Contract Manager, which is the main entry point to Kofax Customer Communications Manager.

Kofax Customer Communications Manager ComposerUI for HTML5 JavaScript API Web Developer's Guide
Describes integration of ComposerUI for HTML5 into an application, using its JavaScript API.

Kofax Customer Communications Manager Batch & Output Management Getting Started Guide
Describes how to start working with Batch & Output Management.

Kofax Customer Communications Manager Batch & Output Management Developer's Guide
Describes the Batch & Output Management scripting language used in CCM Studio related scripts.

Kofax Customer Communications Manager DID Developer's Guide
Provides information on the Database Interface Definitions (referred to as DIDs), which is an alternative method retrieve data from a database and send it to Kofax Customer Communications Manager.

Getting help for Kofax products

Kofax regularly updates the Kofax Support site with the latest information about Kofax products.

To access some resources, you must have a valid Support Agreement with an authorized Kofax Reseller/Partner or with Kofax directly.

Use the tools that Kofax provides for researching and identifying issues. For example, use the Kofax Support site to search for answers about messages, keywords, and product issues. To access the Kofax Support page, go to www.kofax.com/support.

The Kofax Support page provides:

- Product information and release news
  Click a product family, select a product, and select a version number.
- Downloadable product documentation
  Click a product family, select a product, and click **Documentation**.
- Access to product knowledge bases
  Click **Knowledge Base**.
- Access to the Kofax Customer Portal (for eligible customers)
  Click **Account Management** and log in.

To optimize your use of the portal, go to the Kofax Customer Portal login page and click the link to open the *Guide to the Kofax Support Portal*. This guide describes how to access the support site, what to do before contacting the support team, how to open a new case or view an open case, and what information to collect before opening a case.

- Access to support tools

  Click **Tools** and select the tool to use.
- Information about the support commitment for Kofax products

  Click **Support Details** and select **Kofax Support Commitment**.

Use these tools to find answers to questions that you have, to learn about new functionality, and to research possible solutions to current issues.

Chapter 1

Introduction

This chapter describes the syntax and set expressions of the Core scripting language as they are used in CCM Core.

Syntax and conventions of the Core scripting language

Script example: RunMdl script

To understand how CCM Core scripts are organized, you can view an actual CCM Core script call installed with CCM Core.

1. Start CCM Core Administrator.
2. In the tree view, click the **Services** node and select the **Services** tab.
3. Click **RunMdl** and click **Edit script** on the right.
   The Script Editor window appears.
   The script is started with a comment containing a short description of the script.

```
/*
 * RunMdl script. This script is an example that provides
 * default behavior. Use "as is" or adapt to your needs.
 */
```

   The parameter block defines all expected parameters. It specifies defaults for all parameters except `Model`. Parameters that have default values may be omitted when the script is called.

```
Parameter Text Model;
Parameter Text Result = "result.doc";
Parameter Text Keys = "";
Parameter Text Extras = "";
Parameter Text Environment = "";
Parameter Text MetaData = "";
Parameter Text DBB_XMLInput = "";
Parameter Text DBB_XMLOutput = "";
```

   CCM needs to store the result document of the template run in a temporary result file before sending it away. In order to avoid conflicts, the names of these temporary files must be unique. In the following code, this is done by including `_uniqueid` in the file name. `_uniqueid` is a so-called local constant, which provides a unique value every time it is called. For more information on the local constant, see Constants.

```
/*********************************************************
 * Construct a unique filename for the result in the temp dir.
 * We declare the file to be temporary!
 *********************************************************/
```

```
Const Text ResultFile = ("result" + _uniqueid)[TempDir, "doc"];
Temporary File (ResultFile);
```

There is a script command called `ITPRun` that you can use to run templates.

```
ITPRun
      Model (Model)
      Result (ResultFile)
      Keys (Keys)
      Extras (Extras)
      Environment (Environment)
      MetaData (MetaData)
      DBB_XMLInput (DBB_XMLInput)
      DBB_XMLOutput (DBB_XMLOutput);
```

You can use the `SendFile` command to send files to the client application that calls a CCM Core Service. In this case, the result document of the template run is sent to the client.

```
/*********************************************************
 * Return the result document.
 *********************************************************/

SendFile
      Src (ResultFile)
      Dest (Result);
```

4. Close the Script Editor and be sure **not** to save any possible changes that you may have made while viewing the script.

Script example: RunDocumentPackTemplate script

In this section, you can view another CCM Core script call installed with CCM Core. You can use the RunDocumentPackTemplate script to create Document Packs from a Document Pack Template.

1. Start CCM Core Administrator.
2. In the tree view, click the **Services** node and select the **Services** tab.
3. Click **RunDocumentPackTemplate** and click **Edit script** on the right.
   The Script Editor window appears.

   The parameter block defines all expected parameters. It specifies defaults for all parameters except `DocumentPackTemplate`. Parameters that have default values may be omitted when the script is called.

```
Parameter Text DocumentPackTemplate;
Parameter Text Result = "documentpack.zip";
Parameter Text Environment = "";
Parameter Text DBB_XMLInput = "";
```

   To create a Document Pack, you need to create a session that is associated with the active job and has a unique storage area. The session is no longer needed when the job ends, so the parameter `Persistent` is set to `false`. The script is started with a comment containing a short description of the script.

```
/*********************************************************
 * Create a Session.
 * The Session can be removed when the Job ends
 * so Persistent is set to false
 *********************************************************/
CreateSession
   Persistent(false);
```

CCM needs to store the result of the Document Pack Template run in a temporary result file before returning it. To allow a user to open multiple results, the names of these temporary files must be unique. In the following code, this is done by including `_uniqueid` in the file name. `_uniqueid` is a so-called local constant, which provides a unique value every time it is called. For more information on the _uniqueid local constant, see Constants.

```
/*********************************************************
 * Construct a unique filename for the result in the temp dir.
 * We declare the file to be temporary!
 *********************************************************/
Const Text ResultFile = ("result_" + _uniqueid)[TempDir, "doc"];
Temporary File (ResultFile);
```

Use the `CreateDocumentPack` command to set up a new Document Pack.

```
/*********************************************************
 * Create a new document pack
 *********************************************************/
CreateDocumentPack Name ("pack");
```

The `ITPRun` command creates the content of the Document Pack based on a Document Pack Template. The information in the template determines what content is created. `OutputMode ("pack")` indicates that the result is a Document Pack. The local constant `_document_pack` points to the Document Pack.

```
/*********************************************************
 * Run the Document Pack Template
 * This creates the content for the Document Pack
 *********************************************************/
ITPRun
      Model (DocumentPackTemplate)
      Result (_document_pack)
      Environment (Environment)
      OutputMode ("pack")
      DBB_XMLInput(DBB_XMLInput);
```

The `ITPRun` command results in a directory tree containing the content of the Document Pack. Use the `SaveDocumentPack` command to write the active Document Pack to a file.

```
/*********************************************************
 * Save the Document Pack to the previously created
 * filename
 *********************************************************/
SaveDocumentPack
       File (ResultFile);
```

Clean up the Document Pack.

```
/*********************************************************
 * Close the Document Pack
 *********************************************************/
CloseDocumentPack;
```

You can use the `SendFile` command to upload the resulting Document Pack file to the client application.

```
/*********************************************************
 * Return the result file.
 *********************************************************/
SendFile
      Src (ResultFile)
      Dest (Result);
```

4. Close the Script Editor and be sure **not** to save any possible changes that you may have made while viewing the script.

Layout of the RunMdl and RunDocumentPackTemplate scripts

To ensure that the RunMdl and RunDocumentPackTemplate services function properly, follow the recommended layout of their scripts:

- Use a comment block to add a title and a short description.
- Start the script with the parameters.
- Place each command on a new line.
- If a command has multiple parameters, put each parameter on a separate line. Indent the parameters to facilitate reading.

Rules for the syntax

A CCM Core script is basically a list of commands preceded by the input, such as a parameter, variables, and constants, needed by these commands. A script is run from top to bottom: what comes first is resolved first.

To control the flow through a script, you can use conditional statements and iterations. These statements allow you to create a script that can do different things based on the result of an expression. The following is a list of the syntax rules that you should use to make a CCM Core script valid and readable:

- White space is ignored, but you can use it to make scripts readable.
- Every statement in a script must be closed with a semicolon.
- Parentheses: any expression can be enclosed in parentheses to affect the order of calculation.
- Parameters, constants, and variables are typed and must be declared. The types are Text, Number, or Boolean.
- Text: a string value must always be enclosed in quotes. File names and file paths are string values.
- Number: a number must lie between +2.147.483.647 and #2.147.483.648. Numbers can also be written as hex numbers. In that case they must start with 0x or 0X.
- Boolean: a Boolean is either True or False (case-insensitive).
- A command consists of a name followed by parameters and closed with a semicolon. Each parameter in a command has a name and a value between braces.
- Functions: CCM Core supports a set of built-in functions that you can use wherever an expression is expected. Functions receive a list of comma-separated parameters as input and return a single value as their result. The parameters are enclosed in braces. The names of the functions are case-sensitive.

Parameter, constant, and variable names must comply with the following rules:

- A name consists of any sequence of alphanumeric characters and the underscore (A-Z, a-z, 0-9 and _) A name cannot begin with a digit.
- The following keywords cannot be used as names: NOT, not, FI, fi, ELSE, else, ELIF, elif, OD, od, TRUE, true, FALSE, false. These keywords are case-insensitive.
- The symbolic names of variables, parameters, and constants must be unique within their script.
- Names are case-sensitive.

Expression	Result	Description
"c:\temp\test.doc"["",]	"\test.doc"	path removed
"c:\temp\test.doc"[,""]	"c:\temp\test."	extension removed
"c:\temp\test.doc"[,]	"c:\temp\test.doc"	nothing happened
"c:\temp\test.doc"["",""]	"\test."	path removed and extension removed
"c:\temp\test.doc"["c:\archive","pdf"]	"c:\archive\test.pdf"	path and extension changed

How to structure your scripts

Best practice for creating scripts is that all of the scripts must be structured in the same way to enhance readability and facilitate troubleshooting. This section describes structuring methods that you should follow when creating a script.

Comment block

Every script starts with a comment block that contains the following:

- Name of the creator
- Creation date
- Name of the script
- Short description of the scripting function
- Intended use, if the script is intended as a script component, in effect as a command
- Description of pecularities, if something is unusual about the script

Parameters block

The following requirements apply for a parameters block:

- The second block in the recommended structure is the parameter block.
- Parameters are the data, passed to the script in the job call, needed for the script to function.
- Parameters must be declared. When parameters are declared, it becomes clear for the reader what is needed to make the script work. Also, in case of troubleshooting you can easily find an error by comparing the list of passed parameters to the list of needed parameters.
- Use descriptive names for the parameters.
- Use comment when you suspect something might not be clear.
- The use of capitals in parameter names is not prescribed in the Core scripting language, but capitalization must be consistent.

Constant block

- A parameter block is followed by a constants block.
- A constant is declared and initialized at the same time, after which its value is fixed.
- Constants can be initialized with the value of a parameter or a variable.

Example

```
Const Text copiedfilename = destparameter["c:\temp",];
```

```
/* The constant copiedfilename is initialized with the path/filename as stored in the
 destparameter. */

Const Number numberofcopies = Var1 + Var2;
/* The total of two Variable values is stored in a constant. */
```

Variable block

The constant block is followed by the variable declaration block. Variables are entities that can contain values. Variables are typed: Text, Number, or Boolean. Variables are used to give meaningful names to values changed in the script.

Script body

After the parameters, the constants, and the variables are declared, you can add the body of the script. This body consists of a list of commands and control statements. As all parameters, constants, and variables now have meaningful names, this part of the script should be easy to create and maintain.

Indentation

You can use indentation to show that parameters belong to a command.

Example

```
Copy
   Src(source_file)
   Dest(copied_file)
   TimeOut(waittime);
```

In this example, `source_file`, `copied_file`, and `waittime` are either parameters, variables, or constants defined or declared at the beginning of the script.

Start a new Service

CCM Core provides Services. Jobs are submitted to a specific Service of CCM Core. The functionality of such a Service is determined by its script. When a Service is created, a script is also created. The name of the Service is the name of the script.

After a script has been created, its parameters must be mapped to call parameters, it must be compiled, and CCM Core must be restarted to activate the Service.

1. Start CCM Core Administrator.
2. In the tree view, select the **Services** node.
3. Select the **Services** tab and click **Add service**.
   The New Service window appears.
4. Enter a name for the new Service and click **OK**.
   The Script Editor opens.
5. Make necessary changes to the script.

   Note This is written with an assumption that the source file exists and that the Server is allowed to write the result file.

   The following is an example script. You can copy it and paste into the required field if you want to practice.

```
Parameter Text source_file;
Parameter Text result_file;

Copy
 src (source_file)
 dest (result_file);
```

6. Click **File** > **Save** and then close the Script Editor.
7. Now you need to map the parameters. Click the new Service under the **Services** node and select the **Service** tab on the right.
8. In the **Script parameter** column, enter the parameters as listed in the script.

   Example If you continue to create the example script, enter the following parameters.

   ```
   source_file
   result_file
   ```

   Each parameter must be on its own line and be spelled exactly as in the script.
9. In the **Value (job parameter)** column, type the parameters in the order in which they are expected in the job call by entering a number preceded by $.

   Example

   ```
   $1
   $2
   ```
10. Click **Save**.
11. Now compile the script. All scripts are compiled at the same time. Click the **Services** node and select the **Services** tab.
12. Click **Compile scripts**.

    If no problem is reported, you receive a notification that no errors are found. If there are errors, read the error report, correct the errors, and compile the scripts anew.
13. Click **Save & Apply** to restart CCM Core.

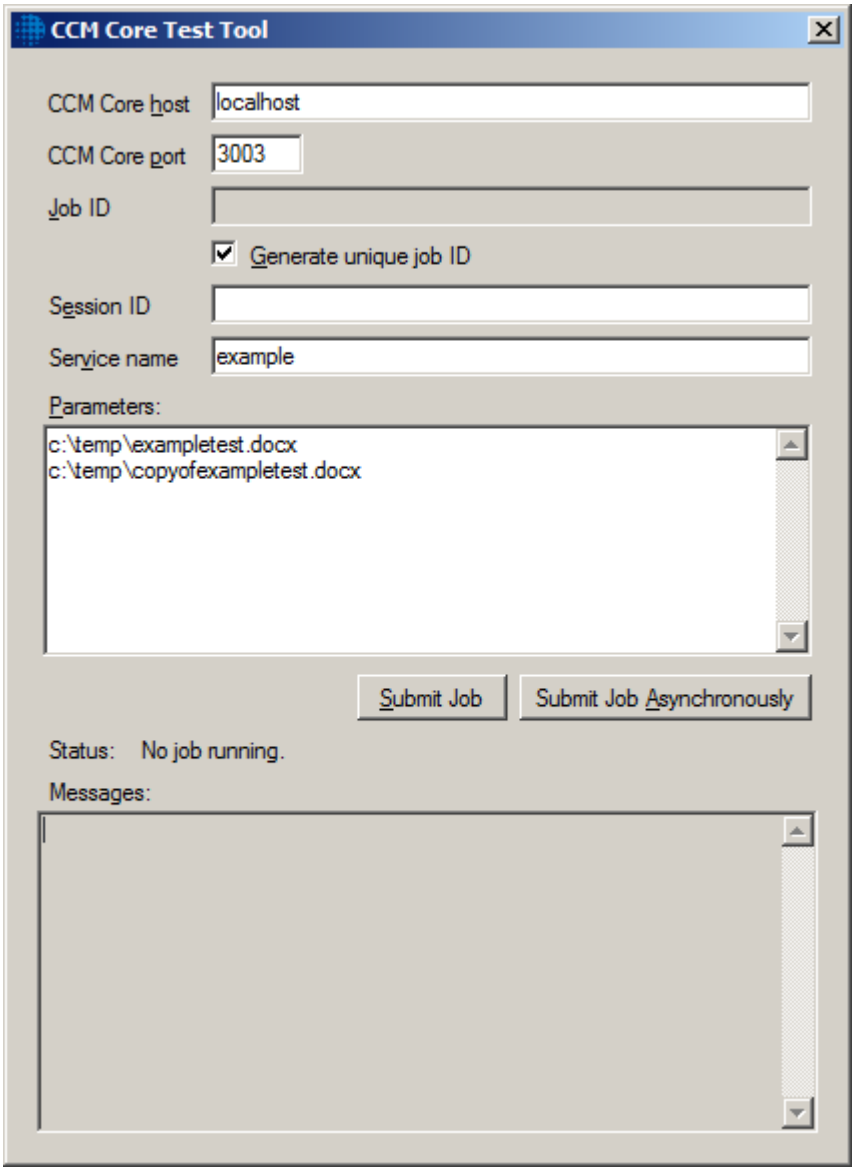
    Now you can use your new Service.

Submit a job to a Service

CCM Core comes with a tool that facilitates submitting jobs for testing. You can use this Test Tool to submit a job to a script. The following procedure describes how you can use the Test Tool to submit a job to the example script that you created in Start a new Service.

1. Create a Microsoft Word file and call it exampletest.
2. Save the exampletest file to `C:\Temp`.
3. Start CCM Core Administrator.
4. On the menu, click **Tools** and click **Test Tool**.

   CCM Core Test Tool opens.
5. Provide necessary information, as shown in the following example. CCM Core host and CCM Core port might differ in your case. The default port for instance 1 is 3003. Session ID can be left empty.



6. Click **Submit**.
   The **Status** pane will inform you when CCM Core is ready.
7. When CCM Core is ready, navigate to `C:\Temp` and locate the copyofexampletest.docx file.
   You have just created and tested a new Service in CCM Core.

Modify file names and file paths

You can modify file names and paths in a CCM Core script.

```
<full file name>[<path>,<extension>]
```

This operation assumes that `<full file name>` is a valid file name. The result of this operation is a valid file name. The name itself is the same as the name of `<full file name>`, but the path is

replaced by `<path>` and the extension by `<extension>`. When an empty string "" is specified, the path or extension are left blank. When `<path>` or `<extension>` are not specified, the path or extension of `<full file name>` is not changed.

Note Only the names are modified, not the files themselves. In other words, by modifying a file name the file itself is not moved on the file system nor is it converted to a different file format.

See the following table for examples.

Expression	Result	Description
"c:\temp\test.doc"["",]	"\test.doc"	path removed
"c:\temp\test.doc"[,""]	"c:\temp\test."	extension removed
"c:\temp\test.doc"[,]	"c:\temp\test.doc"	nothing happened
"c:\temp\test.doc"["",""]	"\test."	path removed and extension removed
"c:\temp\test.doc"["c:\archive","pdf"]	"c:\archive\test.pdf"	path and extension changed

Send email from a script

There is a command and a script command available to send email from within a script. The following section provides additional explanation on how to use the commands.

CCM Core is able to send email through any SMTP server.

The following three types of email messages can be sent:

1. An email containing only text
2. An email containing text and one or more binary attachments
3. A complete, formatted email including all headers and attachments

The email sent by CCM Core should be readable by any MIME-enhanced email client.

Due to the nature of the SMTP protocol, the header of the email is split in two components:

1. The sender and recipient information must be provided to the `Mail` command (see Mail).
2. The other information in the email header, such as Subject, Reply-To, and so on, must be provided in an ASCII text file.

Send email without an attachment

If no attachments are to be sent, the file indicated by the File parameter contains the body of the email document. Any additional headers can be added to the beginning of this file conform the specifications in RFC 822.

Separate the headers from the body of the email message by a blank line.

Example

```
Subject: Your request dated 15 Jan 2018
```

```
To: "F Gonzales" <F.Gonzales@host.domain.com>
Reply-To: "Web Office" <Internet@host.domain.com>

Dear Mr. Gonzales,
..........
Regards,
I. T. P. Server III
```

Note The SMTP server does not use the information in the email to determine the recipient. It is only provided for the convenience of the recipient. The `From` and `To` information is passed to the `Mail` command directly.

Note Any 8-bit characters in the email are converted to 7-bit characters using MIME enhancements. Any MIME enhanced email client displays these characters correctly.

Send email with an attachment

If the `Mail` command has an Attachments(...) parameter, CCM Core sends all the files specified in this parameter as attachments using MIME enhancements.

TXT and HTML attachments

MIME enhanced email can have two alternative bodies displayed based on the preferences of the recipient.

The first two files listed in the parameter Attachments are sent as alternative bodies.

CCM Core sends files listed in the Attachments parameter as alternative bodies based on the following rules:

- If the name of the first attached file has the extension TXT and the second file has the extension HTML, CCM Core sends the first two attachments as alternative versions of the body of the MIME message. All additional files are sent as regular attachments.
- If the name of the first attached file has the extension HTML and the second file has the extension TXT, CCM Core sends the first two attachments as alternative versions of the body of the MIME message. All additional files are sent as regular attachments.
- If the name of the first attachment file has the extension TXT and the second file does not have the extension HTML, CCM Core sends the first attachment as the body of the MIME message. All additional files are sent as regular attachments.

HTML inline images

You can use inline images in the HTML body of the email. This body consists of two parts: one is a tag in the HTML pointing to the image to be shown. The other part is part of the header. This is the content ID.

HTML tag

In the body, use the following tag.

```
 <img src = "cid:image@hostname">
```

Replace image with the actual image name and `hostname` with the actual host name. The `image@hostname` location is the same as used in Attachment parameter of the `Mail` command. The name of the location must be unique for an object.

Content-ID in `MAIL` command

The content-ID placed in the HTML body must also be passed to the `MAIL` command. This is done as part of the Attachments parameter.

```
Attachments("c:\mail\body.txt,c:\mail\body.html,[image@hostname]path\image.gif")
```

Specify the File parameter

While the first two attached files are used as alternative bodies, the file passed with the parameter File to the command `Mail` is used to contain human readable headers and a text displayed when the mail client of the recipient does not have MIME enhancements.

Headers listed in the file specified with the parameter File(...) of the command `Mail` are added to the header of the email.

These are human readable headers:

```
Subject:  E-mail generated with the Mail command within ITP/Server.
To: "S. Kiddy" <s.kiddy@serverscripts.com>
Reply-to: "Our office" <scriptsRus@itpserver.com>.
```

There are other headers that you can use. For a description, see the RFC822 document available on the Internet.

Note You should separate the headers from the body with a blank line.

The **body** of this file is also sent, but is only shown if the recipient has an email client that does not support MIME enhancements. This file usually contains a message informing the recipient that MIME support is required to read the attachments.

See Examples of the Mail command for a sample MIME message.

Error conditions

The `Mail` command fails if the mail server reports an invalid or empty recipient. In that case, the mail message is not sent.

To send email to a list of recipients, use a separate `Mail` command for each recipient to handle possible errors.

The `Mail` command only validates recipients against the local mail server. It does not catch errors such as non-existent remote addresses so the validation does not guarantee that a message is delivered at the destination.

Examples of the Mail command

The following are examples of the `Mail` command used in a script. For a description of the `Mail` command, see Mail.

```
Mail
```

```
    File("c:\mail\mime.txt")
    Host (smtp_server)
    From (sender@somewhere.com)
    To (somebody@somewhere.com, somebodyelse@somewhere.com)
    Attachments("c:\mail\body.txt,c:\mail\body.html,
                 [logo@aia.nl]c:\images\logo.gif,
                  [signature@aia.nl]c:\images\signature.jpg"),
                   c:\mail\invoice.doc,c:\mail\replyletter.doc");
```

A sample content of the mime.txt file.

```
Subject: Your request dated 18 Jul 2002
To: "F Gonzales" <F.Gonzales@host.domain.com>
Reply-To: "Web Office" <Internet@host.domain.com>

This mail is meant to be read using a MIME-enhanced e-mail client.
If you can see this text your client is NOT MIME-enhanced.
Please ask your system operator for assistance.
```

A sample content of the body.txt file.

```
Dear Mr. Gonzales,
......
Please find the following documents attached:
- Your invoice.
- A reply letter.
......
```

A sample content of the body.html file.

```
<html>
<img src = "cid:logo@aia.nl" align="right" border="0">
Dear Mr. Gonzales,<p>
..........
Please find the following documents attached:<br>
<ul>
<li> Your invoice.</li>
<li> A reply letter.</li>
</ul>
.......
With kind regards,
<img src = "cid:signature@aia.nl" border="0">
</html>
```

Any email client that is MIME enhanced shows the header lines from the mime.txt document and displays the content of either the body.txt or body.html file as the body of the email. The choice of which version of the body is shown depends of the preferences of the recipient. A non-MIME enhanced email client shows the MIME message from mime.txt.

MIME types for attachments are read from the Windows Registry.

Send preformatted email

If the preformatted parameter of the `Mail` command is set to True, the document provided with the File(<text>) parameter is considered to contain a complete, formatted email, including all headers and attachments. In this case, the value of the Attachments(<text>) parameter is ignored.

Composing a preformatted email requires an in-depth knowledge of the MIME formatting specifications and exceeds the scope of this guide.

CCM Core sessions

A session is a storage area into which a script can store persistent data across calls to CCM Core. This section provides an overview of how sessions work, and how they are used in scripts.

The provided examples do not discuss all possible options for the commands involved.

Create a session

Usually, when a CCM Core job is started, it is not associated with a session. A script can choose to create a new session using the command `CreateSession`.

```
CreateSession;
```

When the command `CreateSession` completes, the running job is associated with a newly created session, and CCM Core generates the following:

- Session identifier (available through the constant _sessionid)
- Session directory under the instance Work directory that resides in `[drive:]\CCM` (available through the constant _sessiondir)

The script can store persistent data into the session directory, such as an XML file received from the client application.

```
ReceiveFile
  Src("data_for_the_session.xml")
  Dest("data.xml" [_sessiondir,]);
```

This example requests a file from the client application and stores it in the file `"data.xml"` in the session directory. It is also possible to store single string values into the session. For example, this `SetSessionParameter` command stores the value `"3.14"` in the session parameter `my_stored_info`.

```
SetSessionParameter
  Par ("my_stored_info")
  Value ("3.14");
```

The stored data remains available until the session is explicitly cleaned up. To gain access to the session later, it is important to store the session identifier, which is the key by which the session is accessed. You can do so by sending the session identifier to the client application with the function `exchange_data`.

```
Var Text Dummy = exchange_data ("SessionID", _sessionid, 0);
```

The client application must store the session identifier appropriately, so that it can pass it to subsequent CCM Core jobs that require access to the data from the session.

Templates can read session parameters using the `session_parameter` function. Templates cannot change session parameters, but they can use the _Document Field Set instead to pass data back to the CCM Core script that can access these using the `document_metadata` function. For more information on the `document_metadata` function, see document_metadata (k).

Access an existing session

To allow a CCM Core job to access to an existing session, the session identifier must be passed from the client application to CCM Core when the job is submitted. This is similar to how the job ID is specified. The called job is then associated with the session that was specified, and the scripts are able to access the data stored in the session. You cannot associate with an existing session a job that is already running and not yet associated with any session. For more information on how to pass a session identifier to CCM Core when submitting a job, see the chapter "Integration" in the *Kofax Customer Communications Manager Core Developer's Guide*.

When a script requires access to a session, you should always check that it is actually running in a session. To do so, test the value of `_session_active`, as shown in the following example.

```
If Not _session_active Then
  ThrowError Message("This service requires a session.");
Fi;
```

Note When a client application passes a session identifier to CCM Core that does not correspond to an existing session, the job proceeds to run normally. No error is generated. You should always include a check for an active session at the beginning of each script that requires the use of a session.

If the script is running in a session, it can use the data from the storage directory of the session, such as the file `"data.xml"` that was stored in the preceding example.

```
ITPRun
  Model("mymodel")
  Keys( "data.xml" [_sessiondir, ] );
```

The script can also retrieve string values from stored session parameters, as shown in the following example.

```
Var Text X = get_sessionparameter ("my_stored_info");  # returns "3.14"
```

CCM Core does not simultaneously run multiple jobs submitted with the same session identifier. If two jobs are simultaneously submitted with the same session identifier, they run in a series. The reason for this behavior is to prevent simultaneous access to the same session storage by multiple scripts. Such simultaneous access may lead to conflicts.

This serialization also takes place when the session identifiers of the submitted jobs do not correspond to valid sessions. This means that passing a fixed, non-empty session ID to jobs is currently not recommended, because the jobs are not able to use the resources of more than one Document Processor.

Clean up a session

When a script is running within a session, it can clean up that session by calling the `CleanupSession` command.

```
CleanupSession;
```

When this command completes, the data for the session is removed from the disk and the constant _sessiondir becomes empty. Also, the value of _session_active is set to False, and _sessionid is empty.

In addition to this command, you can use the command `ExpireSessions` to clean up old sessions in bulk, based on age (time since session creation) or the elapsed time since the last time the session was accessed.

Save and restore a session

You can archive and later restore the contents of a session. To save the contents of a session, use the command `SaveSession`.

The following example shows how the current session is saved to the archive file `savedsession1` in the directory `C:\SavedSessions`. The stored information includes the contents of the directory _sessiondir and the stored session parameters.

```
SaveSession
 Archive( "savedsession1" [ "c:\SavedSessions", "ses" ] );
```

After the command `SaveSession` completes, the session contents are stored in the specified archive. Furthermore, the session is cleaned up. This means that it is not possible to archive a session and then to continue working within that same session.

To restore the contents of an archived session, use the command `RestoreSession`. The following example demonstrates how the contents of the session that was saved in the preceding example are restored.

```
RestoreSession
   Archive( "savedsession1" [ "c:\SavedSessions", "ses" ] );
```

You can only use the command `RestoreSession` when the current job is not already associated with a session. The command creates a new session and associates it with the current job. It then restores the contents, such as files and session parameters, of the archived session into the newly created session. The new session is created with a new session identifier, not equal to the original session identifier of the session that was saved. After the command `RestoreSession` completes, the original session identifier of the restored session can be retrieved from the variable _restored_session. The variable _sessionid contains the new session identifier.

The archive file is compressed and its contents are optionally encrypted to protect any content included from the directory _sessiondir. A checksum is used to validate the contents before restoring a session.

Applications can add data to the end of an encrypted archive file without invalidating its contents. Non-encrypted archives should not be modified.

CCM ComposerUI

CCM ComposerUI uses the session to store answers to interactive screens and (optionally) cached data. This information is included when the session is saved or restored. When implementing a save/restore scenario, consider whether CCM ComposerUI caching should be enabled and possibly remove any cache files when restoring a session.

Clone a session

You can create an identical copy (clone) of a session using the command `CloneSession`. The following example copies the current session including the contents of the folder _sessiondir and the session parameters.

```
CloneSession;
```

The new session is created with a new session identifier. After the command `CloneSession` completes, this new session identifier can be retrieved from the variable _new_session. The script continues to run in the original session context.

CCM ComposerUI

CCM ComposerUI uses sessions to store answers to interactive screens and (optionally) cached data. This information is included when the session is copied. When implementing a scenario where sessions are copied, consider whether CCM ComposerUI caching should be enabled and possibly remove any cache files when restoring a session.

CCM Document Packs

A Document Pack is a collection of related documents and alternative formats of these documents. The documents are combined in a directory structure with a manifest describing the relationship between the documents.

As of CCM Core version 5.1, commands are available to manipulate the contents of Document Pack directly and track their relationship.

This section provides an overview of how Document Packs work in the context of the CCM Core Scripting language. The provided examples do not cover all possible options for the involved commands.

Create a Document Pack

CCM Core manipulates Document Packs in the context of a session. The session context is used to store the contents of the Document Pack as a directory tree.

Each session can have one active Document Pack. All operations manipulate the contents of this active Document Pack. It is possible to store multiple Document Packs in a session, and activate/deactivate them as required. If a session is closed or expired, all Document Packs are cleaned up as well.

A script sets up a Document Pack using the command `CreateDocumentPack`. See Create a Document Pack.

```
CreateDocumentPack Name(…);
CreateDocumentPack Name(…) LoadFrom(…);
```

When the command `CreateDocumentPack` is completed, the session has an active Document Pack and CCM Core sets up the following:

A directory structure within the session identified by the `Name` (available through the constant _document_pack).

If a Document Pack file is specified through the `LoadFrom` parameter, the file is unzipped in the directory and the contents are validated to confirm that the file is a Document Pack.

If the directory already exists within the session, the contents of the directory structure are imported, reactivating the Document Pack.

Build a Document Pack

The `ITPRun` command builds a Document Pack when it executes a Document Pack Template (see ITPRun). You can use the `OutputMode` parameter to explicitly convert the result of a Document Template into a Document Pack.

The following example produces a Document Pack.

```
CreateDocumentPack Name ("pack")
ITPRun
     Model (template_or_documenttemplate)
     Result (_document_pack)
     OutputMode ("pack");
SaveDocumentPack
     File ("mydocumentpack.zip");
CloseDocumentPack;
```

Manipulate content of a Document Pack

The content of the Document Pack is based on slots in the Document Pack Templates and channels which define exceptions for content within a slot. Each of the documents in the slots and channels can have additional alternative representations in different formats.

It is not possible to introduce new slots or channels into a Document Pack. The only operations allowed are the introduction of new alternative formats and manipulation of existing files.

To locate a file in the Document Pack, use the `get_document_from_pack` function (see get_document_from_pack (s, c, t, d)). This function returns the fully qualified name to the specific document in the pack. The contents of this document can be changed or used for further processing.

The command `IterateDocumentPack` allows the script to walk through the contents of the Document Pack, and call a processing script for each document that matches the filter conditions. See IterateDocumentPack.

The command `InsertDocumentPack` allows the script to insert new alternative representations into the Document Pack or to point an alternative representation to a new file. See InsertDocumentPack.

The following example enumerates all templates in a Document Pack and adds a PDF conversion of each result document to the Document Pack.

```
IterateDocumentPack
     Script (AddPDFAlternative)
     Context ("custom context-sensitive information")
     Status ("T");
```

The script AddPDFAlternative.dss performs the actual conversion.

```
Parameter Text Document;
Parameter Text Slot;
Parameter Text Channel;

If get_document_from_pack (Slot, Channel, "pdf", False) = "" Then
     DocToPDF
          Src (Document)
          Dest (Document [ , "pdf" ]);
     InsertDocumentPack
```

```
          Document (Document [ , “pdf” ])
          Type (“pdf”)
          Slot (Slot)
          Channel (Channel);
Fi;
```

Manipulate Document Packs

Each Document Pack is represented by a directory in the session directory.

You can copy the active Document Pack with the command `CopyDocumentPack`. Optionally, the copy can be activated, abandoning the old active Document Pack. Also, you can use the command `CopyDocumentPack` to remove certain content from the copy. However, such a copied Document Pack (with some content being removed) cannot be submitted to CCM ComposerUI for interactive modification. For more information, see CopyDocumentPack.

You can reactivate a Document Pack using the command `CreateDocumentPack` with the name of the directory. Create a Document Pack.

The following examples show the workflow for creating a copy of a Document Pack for manipulation.

Setup the Document Pack.

```
 CreateDocumentPack Name ("pack");
```

Generate content for the Document Pack.

```
 ITPRun … Result (_document_pack);
```

Create a scratch copy and activate it, deactivating the original Document Pack.

```
CopyDocumentPack NewName ("scratch-copy")Switch (True);
```

Perform operations on the copy, then save and clean up.

```
SaveDocumentPack File
("my-modified-copy.zip");
CloseDocumentPack;
```

Reactivate the original Document Pack.

```
 CreateDocumentPack Name ("pack");
```

Clean up a Document Pack

The command `CloseDocumentPack` disassociates the active Document Pack from the session and disposes the directory tree. See CloseDocumentPack.

CCM Core exit point scripts

CCM Core provides a number of exit point scripts that you can modify to provide custom functionality. These scripts are not overwritten during an upgrade.

DailyTask.dss

The script DailyTask.dss is called once every day at midnight local time.

Parameters

```
Parameter Number ScheduledTime;
```

If all Document Processors are busy when the DailyTask is scheduled, it is delayed until a CCM Document Processor becomes available. You can use the parameter `ScheduledTime` to determine the original time it was scheduled.

Sample script

None

For details on the use of this exit point, see "Scheduled jobs" in the *Kofax Customer Communications Manager Core Developer's Guide*.

HourlyTask.dss

The script HourlyTask.dss is called every hour.

Parameters

```
Parameter Number ScheduledTime;
```

If all Document Processors are busy when the HourlyTask is scheduled, it is delayed until a CCM Document Processor becomes available. You can use the parameter `ScheduledTime` to determine the original time it was scheduled.

Sample script

The sample script calls the command `ExpireSessions` to terminate sessions that have been idle for more than four hours.

For details on the use of this exit point, see "Scheduled jobs" in the *Kofax Customer Communications Manager Core Developer's Guide*.

UserStartUp.dss

The script UserStartUp.dss is called during the startup sequence of the CCM Document Processor, before the first job is accepted.

Parameters

None

Sample script

None

UserShutDown.dss

The script UserShutDown.dss is called during the shutdown sequence of the CCM Document Processor. No more jobs are accepted after this script has run.

Parameters

None

Sample script

None

Chapter 2

Parameters, variables, constants, files, and expressions

Read this chapter to learn about essential elements in the Core scripting language, such as parameters, variables, constants, files, and expressions as they are used in CCM Core.

Parameters

Every CCM Core Service script can require parameters. The caller of the script must provide these parameters. The use of parameters allows scripts to be as generic and flexible as possible.

Scripts can have two types of parameters:

- Required. The caller must provide these parameters when calling a script.
- Optional. The caller may provide these parameters when calling a script. If the parameter is omitted in the call, CCM Core assigns it its default value. The default value is set after = in the declaration of the parameter.

A parameter is made optional by stating a value in the declaration (setting a default value), as shown in this example.

```
Parameter Text first_parameter;
  # This is an required parameter: no default value.

Parameter Text second_parameter = "some text value"
  # This is an optional parameter. Its default value is present
  # and set to "some text value".
```

Rules for parameter names are as follows:

- The parameters must be declared at the beginning of a script, before any instructions.
- A parameter name consists of any sequence of alphanumeric characters and the underscore (A-Z, a-z, 0-9 and _).
- A parameter name cannot begin with a digit or an underscore.
- Parameter names are case-sensitive.
- The following keywords cannot be used as parameter name: NOT, FI, ELSE, ELIF, OD, TRUE, FALSE.

  Note NOT, FI, ELSE, ELIF, OD, TRUE, FALSE are case-insensitive.

- The names of variables, parameters, and constants must be unique within their script.
- A parameter type can be Text, Number, or Boolean.

The parameters have the following syntax.

```
Parameter Type name;
Parameter Type name = default;
```

Note A parameter is always required, unless it has a default value. So the first parameter in the preceding example is a required parameter, and the second one is optional.

Scripts as commands: passing parameters

A script that calls another script may specify parameters as key (value) pairs. Optional parameters can be omitted. In other words, scripts are called in the same way as predefined commands.

The following is an example of the script `RunITP` being called as a command. Note the parameters.

```
RunITP
 Model("rep:/documenttemplate/InstallationTest/Letter")
 Result("c:\tmp\output.doc")
 Options("/qim");
```

This shows the parameters as declared in the script RunITP.

```
Parameter Text Model;
Parameter Text Result;
Parameter Text Options = "/qi";
Parameter text Configuration ="ITPWORKDIR+"\Config\itpcfg";
```

The parameters `Model` and `Result` are required. They must be specified in the call. The parameters `Options` and `Configuration` are optional. The call overrides the `Options` default of `"/qi"`, and then the default for the parameter `Configuration` is used because the parameter `Configuration` is not present in the call.

Map job request parameters to script parameters

CCM Core passes the parameters in a job request to the corresponding Service script in the order they are listed in the job request. Then, these parameters need to be mapped to the script parameters known by their name. To learn how to map parameters, see Start a new Service.

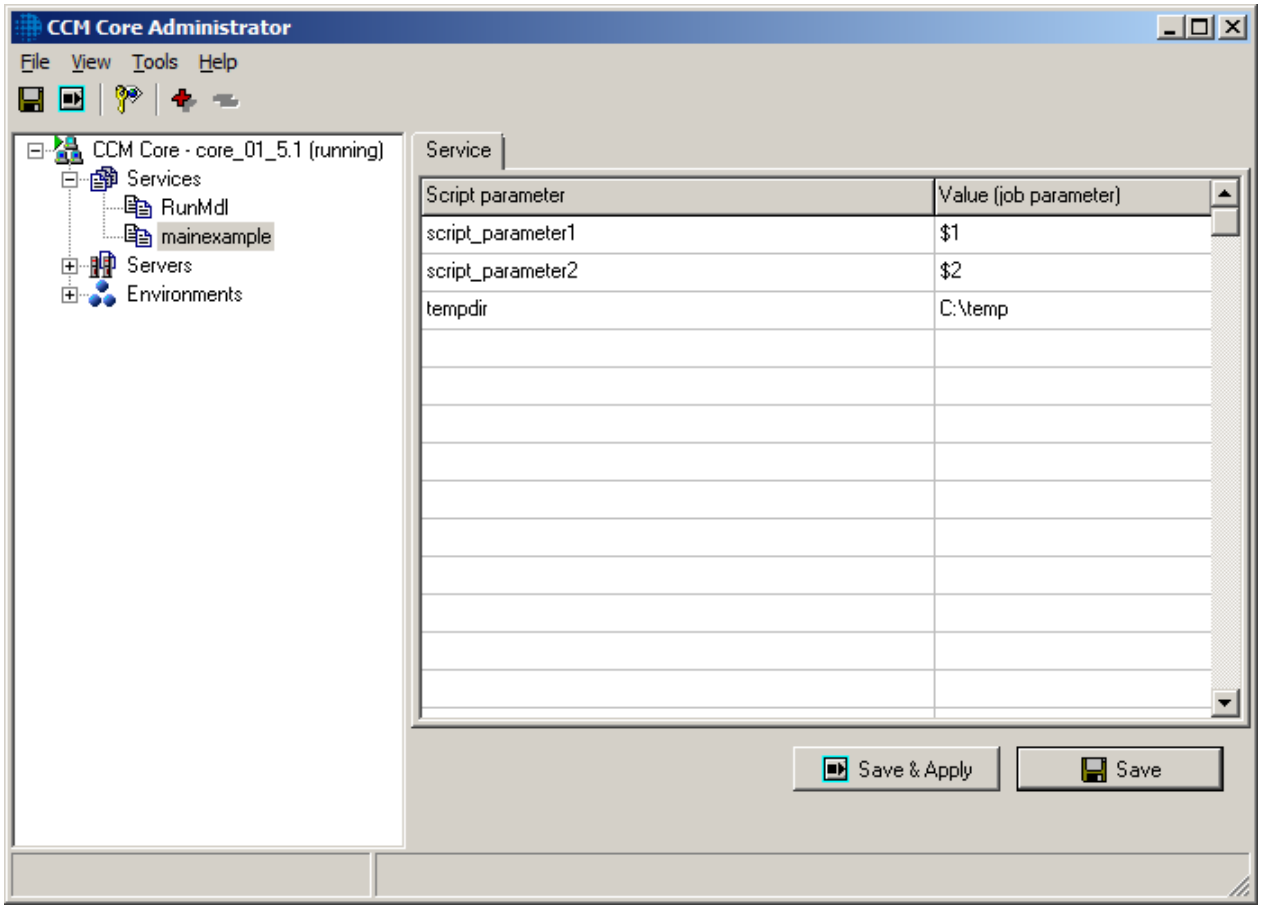
The result of the mapping appears in the dp.ini file of the CCM Core folder `{path}\core\Config`.

Evaluation of expressions proceeds in the following way:

- Each call must start with the name of the Service required.
- Each expression $ n is replaced by the nth parameter specified in the call. If the call does not have an nth parameter, the value of the expression becomes empty.
- Other expressions are passed unmodified.

If an expression passed, such as the actual value passed in the job request, evaluates to an empty value, CCM Core treats the corresponding parameter as undefined. This results in an error if the parameter is required.

Example



Note You can set a value instead of a sequence number on the **Service** tab of the Service node in CCM Core Administrator.

Automatic conversion of types

The following table shows how CCM Core converts expressions when processing parameters passed by a client.

From/To	Numbers	Booleans
Strings	String is interpreted as a number. Non-numeric values cause a run-time error.	Y and True are converted to True. N and False are converted to False. All other values cause a run-time error.

Variables

You can use variables to store the result of an expression under a symbolic name. The value of a variable can be modified at any point in a script by assigning a new expression to the variable.

Note The following keywords cannot be used as variable name: NOT, FI, ELSE, ELIF, OD, TRUE, FALSE.

The following rules apply to all variables:

- A variable name consists of any sequence of alphanumeric characters and the underscore (A-Z, a-z, 0-9 and _). A variable name cannot begin with a digit or an underscore.
- The symbolic names of variables, parameters, and constants must all be unique within their script. The symbolic names of variables are case-sensitive.
- You can only use a variable declared in a conditional or iterating statement from its definition until the end of that statement (statement scope).
- You can declare variables everywhere in the script. They must be declared before you can actually use them. If there is an initial value assigned to the variable, it is evaluated at the moment the variable is declared. If there is no value assigned, the variable is initialized using an empty value.

Syntax

```
Var type name = expression;
Var type name;
```

For the first `Var type` you have to assign an initial value, while the second has no initial value. A variable is empty until a value is assigned to it.

Variable types

The declaration assigns a type to the variable and only values with that type can be assigned to the variable. There are three types of variables:

- Text
- Boolean
- Number

Values of type Number in CCM Core must lie between -2.147.483.648 and +2.147.483.647.

Assign a value to a variable

A script can at any point change the value of a previously defined variable by assigning a new expression or value to this variable.

Syntax

```
 name = expression;
```

The expression is evaluated at the moment the assignment is executed.

Example

This is the script body.

```
Var Text name = "My Name";
Var Number mynumber;
Var Boolean yes_or_no = True;
```

This is the remaining script.

```
name = "his name";
mynumber = (12 + 13)/2;
yes_or_no = 1 < mynumber;
```

Variable scope

You can only use a variable declared in a conditional or iterating statement from its definition until the end of that statement (statement scope).

Constants

Constants are expressions evaluated and then stored under a symbolic name. Constants are used for values that must be known by a symbolic name but that are not changed by the script. For example, any paths for files used or generated by the script could be assembled at the start of the script and stored under a meaningful name, as shown in this example.

```
Const Text checkdir = TempDir+"\checkdir";
```

In this example, the constant `checkdir` is set to contain the path to the directory `checkdir` inside the directory contained in the `TempDir` constant. This `TempDir` is a local constant that contains the temporary directory for the Document Processor running the script. For more information, see Location constants.

Note The following keywords cannot be used as constant name: NOT, FI, ELSE, ELIF, OD, TRUE, FALSE. These keywords are case-insensitive.

The following rules apply to all constant names:

- A constant name consists of any sequence of alphanumeric characters and the underscore (A-Z, a-z, 0-9 and _).
- A constant name cannot begin with a digit or an underscore.
- The symbolic names of variables, parameters, and constants must all be unique within their script.
- The symbolic names of constants are case-sensitive.
- If a constant is defined within a conditional or repeating statement, the constant is only accessible from its definition until the end of this statement.

Local constants

Every CCM Core Service Script can declare its own constants. These constants are accessible only within the script.

Syntax

```
Const type name = value;
```

You can declare variables everywhere in the script. They must be declared before you can actually use them. The value of the constant is evaluated at the moment that it is defined. You can only define a constant once in a script.

The declaration assigns a type to the constant and you can only assign values with that type to the constant. There are three kinds of constants:

- Text
- Boolean
- Number

Values of type Number in CCM Core must lie between -2.147.483.648 and +2.147.483.647.

Constants scope

A constant defined in a conditional or iterating statement is only accessible from its definition until the end of this statement (statement scope).

Global constants

You should declare global constants to define settings that apply to a specific CCM Core setup. Some examples: directory paths, printer names, names of remote servers, or user names and passwords for remote servers.

1. Start CCM Core Administrator.
2. In the tree view, click the **Services** node.
3. In the right pane, select the **Constants** tab and define global constants and their values.
   Every script run on this server can access these settings.
4. Click **Save & Apply**.

The following tables describes five global constants that provide defaults for internal commands.

Constant	Function / Value	Used by Command
WFWPrinter	Default printer driver when printing Microsoft Word files	PrintDocument
PDFPrinter	Default printer driver when printing PDF files	PrintDocument
WFWFormat	Default save format when converting Microsoft Word files	ConvertDocument

Global constants are always assumed to be of type Text.

Internal constants

The following constants are defined by CCM Core internally. You can use them in all scripts.

Constant	Value	Type
_date	The current date in "dd-MMM-yyyy" notation. The abbreviation of the month is not localized and always in English.	Text

Constant	Value	Type
_dayofweek	The current day of the week. Sunday = 0, Monday = 1, and so on.	Number
_document_pack	If a Document Pack is associated with the current session, this constant contains the path to the Document Pack content. If no Document Pack is associated with the current session, this constant is empty.	Text
_error	This constant is set to True if the previous command caused an error; False otherwise. _error is set to False if the command succeeds, and to True if the command fails. The program non-zero return code resides in _returncode in the latter case. If the command started by `RunCommand` could not be started, _error is set to True and _returncode is set to 0.	Boolean
_errorlocation	The script in which the last error occurred.	Text
_expired_sessions	This constant contains a comma-separated list of expired sessions. The content is only valid after a successful `ExpireSessions` command.	Text
_filedate	The current date in "yyyymmdd" notation.	Text
_filetime	The current time in "hhmmssuuu" notation.	Text
_hostname	Name of the machine on which the Document Processor accessing this constant is running.	Text
_interface	CCM Core public client interface in "host:port" format.	Text
_invalidchar	The Unicode REPLACEMENT CHARACTER (U+FFFD).	Text
_itpext	Extension of the last result document. This value is set by the command `ITPRun`. This constant is available in CCM Core 4.4 and higher.	Text
_itplog	Contents of the CCM Core ITPLOG file. This value is set by the command `ITPRun`.	Text
_itplogdm	Contents of the CCM Core ITPLOGDM file. This value is set by the command `ITPRun`.	Text

Constant	Value	Type
_itpstop	Contents of the CCM Core ITPSTOP file. This value is set by the command `ITPRun` and contains the value passed to the instruction STOP.	Text
_jobid	The Job Identifier of the current request.	Text
_message	The error message that caused an command `OnError` to be activated. This variable is set by the command `OnError`.	Text
_newline	A line break character (U+000A).	Text
_new_session	This constant contains the session ID of the last session that was created using the command `CloneSession`. The content is only valid after a successful command `CloneSession`.	Text
_onlinefeaturelevel	The maximum CCM ComposerUI Server feature level supported by CCM Core.	Number
_repository_metadata	The metadata associated with the object returned by the last `RetrieveRepositoryObject` command.	Text
_restored_session	This constant contains the original session ID of the last session that was restored using the command `RestoreSession`. The content is only valid after a successful `RestoreSession` command.	Text
_returncode	The following internal commands set _returncode: `RunCommand`, `PrintDocument`, `ConvertDocument`, `RunMacro`, `SpoolFile`, `PStoPDF`, `CreatePDF`, `ConcatPDF`, `MergePDF`, `SecurePDF`, and `ITPRun`. _returncode is set to 0 and _error to False if the command succeeds, and to the program non-zero return code and _error to True if the command fails. If the command started by `RunCommand` could not be started, _error is set to True and _returncode is set to 0.	Number
_script	The current script being executed.	Text

Constant	Value	Type
_server	The Microsoft Windows Service name of the Document Processor running the script accessing this constant.	Text
_service	Text "Load Balancer Interface:" followed by the name of the CCM Core Service accessing this constant.	Text
_session_active	This constant is set to True if the job is associated with a session; False otherwise.	Boolean
_sessiondir	The storage directory for the current session. This variable is empty if the request is not associated with a session.	Text
_sessionid	If the current job is associated with a session, this constant contains the session ID of that session. If the current job is not associated with a session, this constant is empty.	Text
_servername	Name of the CCM Core installation.	Text
_stderr	"Standard error" output of the last `RunCommand`.	Text
_stdout	"Standard output" output of the last `RunCommand`.	Text
_tab	A TAB character (U+0009).	Text
_time	The current time in "hh:mm:ss" notation.	Text
_uniqueid	This constant returns a new unique text string every time it is used. This constant is useful for generating unique file names.	Text
_user	Owner of the request.	Text
_version	The version of CCM Core.	Text
_version_windows	The version of Microsoft Windows installed on the server hosting the CCM Document Processor.	Text
_version_word	The version of Microsoft Word installed on the server hosting the CCM Document Processor. If Microsoft Word is not installed or CCM Core failed to determine the version, this value is empty.	Text

Location constants

The following locations are defined within CCM Core. You can use them in scripts to write data to or retrieve data from.

Constant	Value	Type
TempDir	Temp location created per Document Processor.	Text
LogDir	Log location created per Document Processor.	Text
InstallDir	The directory where all CCM Core programs are installed.	Text
ITPWorkDir	This is the instance Work directory of a CCM Core setup.	Text

Temporary files

Most scripts create and remove a number of temporary files. CCM Core is able to manage these files automatically in the following way:

- Existing temporary files are removed at the start of the script to prevent conflicts with the files that the script creates.
- At the end of the script or the job all temporary files are removed.

To declare a file temporary, use the command `Temporary`. For more information on the command, see Temporary.

Expressions

CCM Core allows the use of expressions everywhere a value can be used in a script:

- As the value part of a parameter
- As the default value of a parameter or variable
- As the value of a constant
- As the value of a variable during declaration and assignment

The expressions are evaluated and then replaced by their value.

Basic expressions

You can use the following items in an expression:

- Parameters
  A parameter in an expression is replaced by the value of the parameter.
- Constants

A constant in an expression is replaced by the value of the constant.

- Variables
  A variable in an expression is replaced by the current value of the variable.
- Strings
  A string is a sequence of characters between quotes. Use "" to insert a single " in the string.
- Numbers
  A number can be written in one of the following notations: Decimal: as a sequence of digits. Hexadecimal: as a sequence of hexadecimal digits (0-9, a-f), prefixed with "0x." **Example** The number 42 is written as 42 in decimal or 0x2a in hexadecimal.
- Booleans
  You can use Boolean values in expressions where a conditional value is expected. The Boolean values are True and False.
- Parentheses
  You can enclose any expression in parentheses to affect the order of evaluation.

Operations in expressions

CCM Core supports the following operations in expressions.

Operation	Description	Operands	Result	Remarks
x Or y	Logical or	Boolean	Boolean	CCM Core uses short-circuit evaluation. If x evaluates to True, y is not evaluated.
x And y	Logical and	Boolean	Boolean	CCM Core uses short-circuit evaluation. If x evaluates to False, y is not evaluated.
x = y	Equal		Boolean	Comparisons on strings are done case-sensitive.
x <> y	Not equal		Boolean	
x <= y	Less or equal		Boolean	
x < y	Less		Boolean	
x >= y	Greater or equal		Boolean	
x > y	Greater		Boolean	
x + y	Add	Number	Number	The function of the "+" operator depends on its operands.
x + y	Concatenate	String	String	
x - y	Subtract	Number	Number	
x * y	Multiply	Number	Number	

Operation	Description	Operands	Result	Remarks
x / y	Divide	Number	Number	
x % y	Modulo	Number	Number	
Not x	Negation	Boolean	Boolean	
-x	Negation	Number	Number	

Priorities

When multiple operators occur in an expression, the expression is evaluated in a specific order depending on the operators in the expression. You can use a set of parentheses to change the order of expression evaluation.

The following table shows the priority of the operators used in expressions.

Priority	Operators
1	- (Negation), Not (Negation)
2	* (Multiply), / (Divide), % (Modulo)
3	+ (Add), + (Concatenate), - (Subtract)
4	<= (Less or equal), < (Less), >= (Greater or equal), > (Greater)
5	= (Equal), <> (Not equal)
6	And (logical and)
7	Or (logical or)

A priority of 1 is the highest priority (signed values are evaluated first); a priority of 7 is the lowest priority (Or operations are evaluated last). When operators with different priority levels appear in an expression, operations are performed according to priorities.

When operators of the same priority appear in an expression, operations are performed from left to right within the expression. You can always use parentheses to control the order in which operations are performed. The value of a parenthetical expression is determined from the lowest level to the highest level, following the priority rules within matching sets of parentheses.

Example

```
1 + 1        # results in 2.
"1" + "1"    # results in "11": two text values concatenated.
(10+15)/5    # results in 5: 10+15=25, 25/5=5.
10+15/5      # results in 13: 15/5=3, 10+3=13.
```

Comments

Within a CCM Core script, you can use two comments styles.

Line comment

Line comments start with a hash. Everything from the hash symbol up to the end of the line is ignored.

Syntax

```
# Followed by comment
```

Block comment

Block comments are enclosed within a `/* … */` pair and can span multiple lines. Block comments can be nested.

Syntax

```
/*
This is a multi line /* nested */ block comment.
*/
```

Chapter 3

Commands

A command consists of a name followed by parameters and closed with a semicolon. Each parameter in a command has a name and a value between braces. This chapter provides a description of the built-in CCM Core commands. The commands are listed in alphabetical order.

In CCM Core, you can also use a script as a command in another script.

Scripts as commands

In CCM Core you can use every script as a command in another script. These scripts are also called script components. When a script is used as a command, its parameters become the parameters of the command.

Examples of script components

The commands `ChangeBins`, `CreatePath`, `SetSessionParameter`, `SimpleMail`, and `SubmitMaintenanceJob` are actually implemented as script components, which are included in the built-in CCM Core scripts library.

The script code for these script components is included with examples that reside here: `<deploy root>\CCM\Documentation\5.2\Resources\Examples\Core Scripting`. CCM Core does not use these scripts to implement the commands, they are provided as examples only.

When using these example scripts as the basis of your own script components, rename the scripts. If the scripts are not renamed, CCM Core does not use them, because they have the same names as those of the built-in script components, and the built-in script components take priority over user-defined script components.

Removed support

The support for the following commands has been removed:

- `HTMLToPDF`
- `ImageToPDF`

AttachFilesToPDF

Use this command to add one or more files as attachments to a PDF file.

Syntax

```
AttachFilesToPDF
   Src (<text>)
   Dest (<text>)
   Attachments (<text>)
   TimeOut (<number>);
```

Parameters

- `Src`: Required. Source PDF file.
- `Dest`: Required. Result PDF file containing the source PDF with the attached files.
- `Attachments`: Required. One or more file names, separated by commas. You can specify an empty string here, or an empty entry before or after a comma. Also, you can override the name of the attachment by adding a prefix `[name]` to the attachment file name. Verify that the file name refers to an existing file.
- `TimeOut`: Optional. Timeout for this command in seconds. If it is exceeded, the process is terminated and CCM Core reports a run-time error. To disable the timeout, set this parameter to 0 or omit it.

Note the following when using the command:

- The files specified in `Src` and `Dest` must be different files.
- If the source PDF file already contains attachments, they are removed from the result PDF file.

Example

```
Const Text src = "letter" [ _sessionDir, "pdf" ];
Const Text dest = "extended_letter" [ _sessionDir, "pdf" ];

AttachFilesToPDF
   Src (src)
   Dest (dest)
   Attachments ("c:\documents\policy.docx ,
               [Terms and Conditions.docx] c:\documents\tac.docx");
```

In this example, a PDF document named `extended_letter` is created for the source PDF document named `letter` and gets the following DOCX attachments: `policy` and `Terms and Conditions`. The source document for *Terms and Conditions* is the `tac.docx` file. The original file name is overridden by the prefix `[Terms and Conditions.docx]`.

ChangeBins

Call the ChangeBins tool to change the paper tray settings in a Microsoft Word DOC or Microsoft Word DOCX document.

You can use the ChangeBins tool to change the paper trays in a document from one printer configuration to another printer based on a configuration file that provides a mapping between printer (type)s. For more information, see ChangeBins tool.

Syntax

```
ChangeBins
   Document(<text>)
   From(<text>)
   To(<text>)
   Overrides(<text>);
```

Parameters

- `Document`: Required. The document to be changed.
- `From`: Required. The printer (type) the document was originally configured for.
- `To`: Required. The printer (type) the document is to be configured for.
- `Overrides`: Optional. Additional overrides on the mapping.

The `ChangeBins` command requires that a ChangeBins.cfg file is present in the Tools directory. This file must contain mappings for both the `From` and `To` printers.

CleanupSession

The command `CleanupSession` instructs CCM Core to close the session associated with the current job and clean up all resources stored in the session directory. For more information on sessions, see CCM Core sessions.

Syntax

```
CleanupSession;
```

The command `CleanupSession` removes the directory structure created by the command `CreateSession`. After the command `CleanupSession` completes, the constant _sessiondir becomes empty, the constant _session_active is set to False, and the constant _sessionid becomes empty. The command `CleanupSession` fails if the active job is not associated with a session.

CloneSession

CCM Core clones a new session which contains the state of the session associated with the job and a copy of all files stored in the folder_sessiondir and its subfolders. For more information on sessions, see CCM Core sessions.

The command `CloneSession` fails if the active job is not associated with a session.

Syntax

```
CloneSession;
```

Parameters

The command `CloneSession` has no parameters.

The command `CloneSession` saves the session parameters and copies all files and folders from the folder _sessiondir. If applications running in the background have a lock on one of the files in the folder _sessiondir, the command may fail.

If the command `CloneSession` completes successfully, the variable _new_session contains the session identifier of the newly created session.

CloseDocumentPack

Use the command `CloseDocumentPack` to discard the active Document Pack from the session.

The command `CloseDocumentPack` fails if there is no active Document Pack in the session.

Syntax

```
CloseDocumentPack
 KeepTree (True or False);
```

Parameters

`KeepTree`: Optional. Indicates whether the content from the Document Pack should be removed from the session folder. If this parameter is not specified, the content is removed.

If content is not removed from the session folder, the Document Pack can be activated again using the `CreateDocumentPack` command.

ConcatPDF

`ConcatPDF` concatenates two PDF documents.

Syntax

```
ConcatPDF
   File1(<text>)
   File2(<text>)
   Dest(<text>)
   ProducePDFA(<True or False>);
   Processor("Amyuni" or "PDFLib");
```

Parameters

- `File1`: Required. The full path of the first PDF document.
- `File2`: Required. The full path of the second PDF document. This document is concatenated to the `File1` PDF document.
- `Dest`: Required. The full path of the resulting PDF document that contains the concatenated pages. If the source PDF file contains attachments, they are removed from the result PDF file.
- `ProducePDFA`: Optional (Default setting: False). This parameter determines whether the resulting document is PDF/A-1b compliant.
- `Processor`: Optional. Selects the conversion technology used to concatenate the PDF files. If this parameter is set to "Amyuni", the Amyuni toolkit is used. If this parameter is set to "PDFLib", the PDFLib

toolkit is used. Default is "Amyuni". You can change the default through the following setting in the dp.ini file:

```
PDFTools.Processor=<"Amyuni" or "PDFLib">
```

Note The command can only produce PDF/A-1b compliant documents if both input documents are PDF/A-1b compliant and if the parameter ProducePDFA is set to True.

Also, when selecting the conversion technology, note the following:

- **Amyuni**
  - `ConcatPDF` is only supported if both input documents are produced using `DocToPDF` with the Processor value set to "Word".
  - `ConcatPDF` fails if File1 and File2 refer to the same file.
- **PDFLib**
  - `ConcatPDF` can be used to concatenate more than two documents. To concatenate a list of documents, pass the first document as the File1 parameter. Pass the other documents to the File2 parameter as a comma-separated list. Prepend this list with an extra comma to identify the value as a list.
  - `ConcatPDF` cannot transfer interactive elements and document structure information from the source documents.
  - As of CCM Core version 5.1.1, hyperlinks and bookmarks are transferred to the result.

Note When a PDF document is concatenated to a secured PDF document, the following result is obtained, depending on the selected conversion technology:

- For Amyuni, the resulting PDF document has the same security settings as the PDF document specified in the File1 parameter.
- For PDFLib, the resulting PDF document has no security set.

Global settings

The following setting, when added to the [Configuration] section of the CCM Core installation dp.ini file, changes the behaviour of the `ConcatPDF` command. This setting is specific for the PDFLib conversion technology and is ignored when not applicable.

`PDFLib.StrictParameterCheck`

Set to Y to have the `ConcatPDF` command report an error when it encounters an unsupported parameter or setting if `Processor` is set to `PDFLib`.

```
PDFLib.StrictParameterCheck=<Y or N>
```

If not set, it defaults to N.

ConvertCodepage

The command `ConvertCodepage` copies a file and performs a code page conversion.

Syntax

```
ConvertCodepage
   Src(<text>)
   Dest(<text>)
   From(<text>)
   To(<text>)
   SubstituteChar(<text>)
   SubstituteCode(<number>)
   TimeOut(<number>);
```

Parameters

- `Src`: Required. The file with content to be converted.
- `Dest`: Required. The destination of the converted file.
- `From`: Required. Code page in which the file Src is stored. See the table for special values.
- `To`: Required. Code page to which the file Dest has to be converted. See the table in this section for special values.
- `SubstituteChar`: Optional. Replacement character used if a character from the file Src could not be translated into the code page mentioned under To.
- `SubstituteCode`: Optional. Numerical representation for the replacement character used if a character from Src could not be translated into the code page `To`.
- `TimeOut`: Optional. The maximum amount of time in seconds the `ConvertCodepage` command waits for the client to report its code page if the client code page is specified as a From or To parameter. If this time is exceeded, the upload is aborted and an error is reported. If this parameter is omitted, CCM Core uses the default timeout interval as configured in CCM Core Administrator. A value of 0 disable the timeout. .

The parameters To and From specify the source and target code pages. These code pages can be one of the following values.

Value	Description	Source of translation table
"0"	Use the local Windows code page.	Uses Windows code page translation system.
"nnn"	Any number nnn: the number of the code page.	
"client"	Query the remote client for its code page.	Client can either provide a full translation table or a code page number.
"local"	Use the local Windows code page.	Uses Windows code page translation system.
"unicode"	Use Unicode (UTF-16 encoding) in Intel 386 byte order (little endian).	
"unicode-big-endian"	Use Unicode (UTF-16 encoding) in big endian byte order.	
"windows"	Use the local Windows code page.	Uses Windows code page translation system.

If both `To` and `From` specify the same code page, CCM Core copies the `Src` file to `Dest` as if the command `CopyFile` is specified. Also, the code pages " 0," " local," and "windows" all perform the same translation using the local Windows code page. These aliases are provided for backward compatibility.

Note The command `ConvertCodepage` only supports translations for code pages that use a fixed-length character encoding. Code pages that use a variable-length encoding, such as MBCS SI/SO sequences or UTF-8, are not supported. The code page translations "unicode" and "unicode-big-endian" are limited to the UCS#2 subset of UTF-16 (characters U+0000 .. U+FFFF). Characters in the supplementary planes are considered unmappable and replaced with the replacement character.

If the `ConvertCodepage` command is terminated due to a timeout, the connection to the CCM Core process is reset to ensure that data in transit is discarded and processes resynchronize correctly. This forced reset can result in additional network-related errors in the log for this job. Any further communication from this job between the Document Processor and the client fails. The client is informed that the Document Processor is disconnected.

Code page location search order

CCM Core searches for code page translation tables in the following order:

1. Using a custom code page mapping file
2. Using the code page translation tables provided by Microsoft Windows
3. Using built-in code page tables

If CCM Core is unable to locate the appropriate code page translation tables, the conversion fails.

The following code pages are built in CCM Core.

Code page	Belongs to	Code page	Belongs to
273	Germany EBCDIC	912	Latin 2 ISO 8859-2
277	Denmark/Norway EBCDIC	915	Cyrilic ISO 8859-5
278	Finland/Sweden EBCDIC	921	Baltic ISO
280	Italian EBCDIC	922	Estonia
284	Spanish EBCDIC	930	Japan EBCDIC
285	UK EBCDIC	942	Japan SAA (1041+301)
290	Japanese EBCDIC	943	Japan JIS-90 (1041+941)
297	French EBCDIC	1004	Windows
420	Arabic Bilingual	1025	Cyrilic EBCDIC
424	Israel EBCDIC	1026	Turkey ISO 8859-9
437	US	1027	Japanese Latin EBCDIC
500	International EBCDIC	1112	Baltic EBCDIC
813	Greece ISO 8859-7	1122	Estonia EBCDIC

819	Latin 1 ISO 8859-1	1140	Portugal/US EBCDIC with Euro
850	Multilingual	1141	Germany EBCDIC with Euro
852	Latin 2	1142	Denmark/Norway EBCDIC with Euro
855	Cyrilic	1143	Finland/Sweden EBCDIC with Euro
857	Latin 5	1144	Italian EBCDIC with Euro
860	Portugal	1145	Spanish EBCDIC with Euro
861	Iceland	1146	UK EBCDIC with Euro
862	Israel	1147	French EBCDIC with Euro
863	Canada (French)	1148	International EBCDIC with Euro
864	Arabic	1149	Iceland with Euro
865	North European	1250	Latin 2 Windows
866/878	Russia/Russian Internet	1251	Cyrilic Windows
869	Greece	1252	Latin 1 Windows
870	Latin 2 EBCDIC	1254	Turkey Windows
871	Iceland	1257	Baltic Windows
874	Thailand	1275	Apple Latin 1
897	Japan		

The parameter `SubstituteChar` provides a default character used if a character from the `Src` file could not be mapped to `Dest`. If the parameter `SubstituteChar` contains more than one character, only the first character is used.

If the parameter `SubstituteChar` is not present or if it contains an empty string, the value of the parameter `SubstituteCode` is used as the numeric representation of the replacement character.

Note The parameter `SubstituteChar` is ignored if built-in translation tables are used for the conversion. In this case a space is used as the substitution character.

If the parameter `To` maps to the code page "unicode" or "unicode-big-endian," you can also use the parameter `SubstituteCode` to specify Unicode characters as replacement. If the parameter `To` maps to a single-byte code page, only the lowest byte of the parameter `SubstituteCode` is used.

Examples are provided in the table.

Value	Description	Example
ddd	Any character code ddd in decimal format.	"8364" maps to the euro symbol € in the Unicode codepages.

0xhhh	Any character code hhh in hexadecimal format.	"0x20ac" maps to the euro symbol € in the Unicode codepages.

A `SubstituteCode` value of 0 is ignored.

Note If neither `SubstituteChar` nor `SubstituteCode` provide a valid replacement character, the command `ConvertCodepage` reports an error if the conversion encountered an character that could not be mapped.

Code page mapping files

The user can provide custom code page mapping files, which should send a mapping from a specific single-byte code page to Unicode.

These files should follow this naming convention: *CP-nnn.cpt*, where *nnn* is the decimal number of the code page. The file should contain a set of mappings:

```
<Single byte code>:  <Unicode equivalent>
```

Each mapping should be on its own line. You can add comments using a hash.

The codes can be specified in decimal notation (default) or hexadecimal notation (prefixed with "0x").

```
# Sample codepage translation table, using hexadecimal encoding
# Format: <single-byte>: <unicode> # <comment>
0x00: 0x0000 # Map NUL to NUL
# ... more lines here.
0x20: 0x0020 # SPACE
0x21: 0x0021 # !
0x22: 0x0022 # "
0x23: 0x0023 # #
0x24: 0x0024 # $
0x25: 0x0025 # %
0x26: 0x0026 # &
0x27: 0x0027 # '
0x28: 0x0028 # (
# ...
```

Single-byte characters not specified in this file are considered unmappable and cause a translation error if CCM Core encounters them.

Note CCM Core looks for the code page mapping files in the bin directory of CCM Core. You can configure an alternative location by adding the setting CodepageDir=<location> to the [Configuration] section of the dp.ini file.

ConvertDocument

CCM Core instructs Microsoft Word to open the `Src` document and to save this document to the file `Dest` in the specified format. The format of the `Src` document should be one of the document formats that CCM Core recognizes.

You can specify the format for the `Dest` document, depending on the formats supported by Microsoft Word.

Before CCM Core can use Microsoft Word, it must be configured appropriately. For more information, see the section "Use of Microsoft Word by CCM Core" in the *Kofax Customer Communications Manager Installation Guide*.

Syntax

```
ConvertDocument
   Src(<text>)
   Dest(<text>)
   TimeOut(<number>)
   WFWFormat(<number>);
```

Parameters

- `Src`: Required. The word processor document that CCM Core has to convert. Currently only Microsoft Word formats are supported.
- `Dest`: Required. The name of the resulting document. Any existing file is overwritten.
- `TimeOut`: Optional. The timeout for this command in seconds. If the word processor exceeds this timeout, the process is terminated and CCM Core reports a run-time error. If this parameter is omitted, the appropriate default timeout value for either batch or interactive jobs is used. A timeout value of 0 disables the timeout.
- `WFWFormat`: Optional (needed if Src is a Microsoft Word document). The format of the result document if CCM Core uses Microsoft Word to convert the document. If this parameter is not specified, CCM Core uses the value of the constant WFWFormat. If CCM Core uses Microsoft Word and this constant is not set, CCM Core reports a run-time error.

Save formats

Microsoft Word

Specifies the number for the converter to use. The following list specifies the documented converters that are always available. If additional converters are installed, these converters can be enumerated using a macro.

The following formats are documented for Microsoft Word:

- 0 (wdFormatDocument): Microsoft Office Word 97-2003 binary file format
- 1 (wdFormatTemplate): Word template format
- 2 (wdFormatText): Microsoft Windows text format
- 3 (wdFormatTextLineBreaks): Windows text format with line breaks preserved
- 4 (wdFormatDOSText): Microsoft DOS text format
- 5 (wdFormatDOSTextLineBreaks): Microsoft DOS text with line breaks preserved
- 6 (wdFormatRTF): Rich Text format (RTF)
- 7 (wdFormatEncodedText): Encode text format
- 7 (wdFormatUnicodeText): Unicode text format
- 8 (wdFormatHTML): Standard HTML format
- 9 (wdFormatWebArchive): Web archive format
- 10 (wdFormatFilteredHTML): Filtered HTML format

- 11 (wdFormatXML): Extensible Markup Language (XML format)

The following formats are documented for Microsoft Word 2007 and later versions:

- 12 (wdFormatXMLDocument): XML document format
- 13 (wdFormatXMLDocumentMacroEnabled): XML document format with macros enabled
- 14 (wdFormatXMLTemplate): XML template format
- 15 (wdFormatXMLTemplateMacroEnabled): XML template format with macros enabled
- 16 (wdFormatDocumentDefault): Word default file format
- 17 (wdFormatPDF): PDF format
- 18 (wdFormatXPS): XPS format

The following formats are documented for Microsoft Word 2010 and later versions:

- 19 (wdFormatFlatXML): Open XML file format saved as single XML file
- 20 (wdFormatFlatXML): Open XML file format with macros enabled saved as a single XML file
- 21 (wdFormatFlatXMLTemplate): Open XML template format saved as a XML single file
- 22 (wdFormatFlagXMLTemplateMacroEnabled): Open XML template format with macros enabled saved as a single XML file
- 23 (wdFormatOpenDocumentText): OpenDocument Text format

The following format is documented for Microsoft Word 2013 and later versions:

- 24 (wdFormatStrictOpenXMLDocument): Strict Open XML document

Example WFWFormat(6) saves a document in Rich Text format (RTF).

For more details on these formats, see the Microsoft Office documentation.

Get a list of available formats and their numbers

The CCM Core distribution provides a macro that yields a list of available formats and their numbers.

1. Start Microsoft Word.
2. Open the file ITP Document Services Tools located in `{path}\CCM\Programs\<version>\Core\Tools`.
3. Run the macro Overview or SaveFormats.Overview.
   The macro creates a document containing an overview of all converters installed and their respective save formats.

Copy

Use the `Copy` command to create a copy of the specified file.

The command fails if the file `Dest` already exists.

Syntax

```
Copy
   Src(<text>)
   Dest(<text>)
```

```
   TimeOut(<number>);
```

Parameters

- `Src`: Required. The file that CCM Core copies.
- `Dest`: Required. The name of the copy that CCM Core creates.
- `TimeOut`: Optional. The maximum amount of time in seconds that CCM Core waits if another process is locking Src. If this parameter is not specified, CCM Core waits indefinitely for the file to be unlocked. If the value of this parameter is 0 and Src is locked, CCM Core immediately reports an error.

The device `\\.\nul` is handled as a special value for the Src parameter. If this device is used as the source of the copy, an empty file is created.

CopyDocumentPack

Use the command `CopyDocumentPack` to create a copy of the active Document Pack. Optionally, you can filter slots and channels while copying.

The command `CopyDocumentPack` fails if no active Document Pack exists in the session or if there is already an object with the indicated name in the session directory.

Syntax

```
CopyDocumentPack
    NewName (<text>)
    Switch (True or False);
    ExcludeSlots (<text>)
    IncludeSlots (<text>)
    Channel (<text>)
    Prune (True or False);
```

Parameters

- `NewName`: Required. Specifies the name for the directory in the session directory that will contain the contents of the Document Pack. The name must be limited to the characters A-Z, a-z, 0-9, underscores, and dots.
- `Switch`: Optional. Indicates whether the copy of the Document Pack should be made active instead of the current active Document Pack. Default is True.
- `ExcludeSlots`: Optional. Comma-separated list of slots in the Document Pack that are *not* copied if they are present in the active Document Pack. If this parameter is omitted or empty, all slots are copied.
- `IncludeSlots`: Optional. Comma-separated list of slots in the Document Pack that are copied if they are present in the active Document Pack. If this parameter is omitted or empty, all slots are copied.
- `Channel`: Optional. Name of the channel that is copied from the active Document Pack. If this parameter is omitted or empty, all channels are copied. You can use the special value `*default` to copy the default channel.
- `Prune`: Optional. Indicates whether files from the active Document Pack that are not associated with a slot or channel are copied. If this parameter is set to False, all objects are copied and only the manifest for the copy is updated. Default is True.

The following applies when using the filters:

- If the `ExcludeSlots`, `IncludeSlots`, and `Channel` filters are omitted, the command `CopyDocumentPack` creates a full copy of the content in the Document Pack, and the `Prune` option is ignored. If any of the options is used, the appropriate filter is applied. Optionally, all files that are not referenced in the manifest.xml file are removed from the copy.
- The `ExcludeSlots` option overrides the `IncludeSlots` option. Slots listed in the `ExcludeSlots` option are always excluded from the copy.
- If the `Channel` parameter is used, the named channel becomes the default channel in the copied Document Pack. If a slot in the active Document Pack does not have a document for the named channel, the document for the default channel is copied instead.

Note When required slots are removed from the Document Pack after filtering the content with the described options, the Document Pack no longer complies with the definition of the Document Pack Template. Therefore, any operations that update the manifest.xml file, such as CopyDocumentPack and SaveDocumentPack, fail and an error is shown.

Also, you cannot submit filtered Document Packs to CCM ComposerUI for interactive modification.

If the copy is not activated immediately, you can activate it later with the command `CreateDocumentPack` (see CreateDocumentPack).

CopyPDF

Use the command `CopyPDF` to copy a set of pages into a new PDF document.

Syntax

```
CopyPDF
 Src (<text>)
 Dest (<text>)
 PageSet (<text>)
 ProducePDFA (True or False);
```

Parameters

- `Src`: Required. The full path of the source PDF document.
- `Dest`: Required. The full path of the result PDF document that contains the selected pages.
- `PageSet`: Optional. Specifies the set of pages that must be copied into the resulting PDF document. If this parameter is omitted or empty, the full document is copied. If the source PDF file already contains attachments, they are removed from the result PDF file.
- `ProducePDFA`: Optional. (Default setting: False). This parameter determines whether the resulting document is PDF/A-1b compliant.

Note The command can only produce PDF/A-1b compliant documents if the source document is PDF/A-1b compliant and if the parameter `ProducePDFA` is set to True.

The set of pages that must be copied is specified as a comma-separated list of page ranges. Ranges refer to physical pages in the document, *not* to page numbers that might appear on the pages.

Ranges can be expressed in the following ways:

- `n` Page *n*.
- `n-m` Pages *n* to *m* inclusive.
- `-m` All pages from the start of the source document up to page *m* inclusive.
- `n-` All pages from page *n* to the end of the source document.
- `!xxx` Exclude range *xxx*. Excluded ranges have priority over included ranges.

If the set of pages only contains excluding ranges, the whole document except those ranges is copied.

All pages are included in their original order. It is not possible to reorder or duplicate pages with the `CopyPDF` command.

Examples

```
CopyPDF ... PageSet (“2-“); Copy all pages except the first page.
```

```
CopyPDF ... PageSet (“!1”); Copy all pages except the first page.
```

```
CopyPDF ... PageSet ("8,!3,5,2-6"); Copies pages 2,4,5,6 and 8
from the source document.
```

`CopyPDF` retains hyperlinks and bookmarks that refer to pages included in the result document. Interactive elements and elements that refer to removed pages are omitted from the result document.

CreateDirectory

Use the `CreateDirectory` command to create a specified directory.

The command `CreateDirectory` succeeds if the specified directory could be created or if the directory already exists. The command only succeeds if the parent directory already exists. Use the script `CreatePath` to create a directory path with a single command.

Syntax

```
CreateDirectory
   Dir(<text>)
   Security(<text>);
```

Parameters

- `Dir:` Required. The directory that CCM Core creates.
- `Security:` Optional. Directory used to set security attributes on the new directory.

The new directory inherits the compression and encryption attributes from its parent directory. If the Security parameter is used, the directory access rights from that directory are applied to the new directory. If there is no Security parameter specified, Windows NT assigns default access rights to the new directory.

The Security parameter of the command `CreateDirectory` only works properly for Microsoft Windows 2000 and higher.

CreateDocumentPack

Use the command `CreateDocumentPack` to create a new Document Pack or activate a Document Pack that was previously copied or closed.

The command `CopyDocumentPack` fails if there is no active session or if there is already an active Document Pack in the session.

Syntax

```
CreateDocumentPack
      Name (<text>)
      LoadFrom (<text>);
```

Parameters

- `Name`: Required. Specifies the name for the directory in the session directory that will contain the contents of the Document Pack. The name must be limited to the characters A-Z, a-z, 0-9, underscores and dashes. If the directory already exists, the content is treated as an unpacked Document Pack file.
- `LoadFrom`: Optional. Import the contents from a file into the Document Pack. This file must have been created using the command `SaveDocumentPack`.

The command `CreateDocumentPack` supports the following scenarios:

- Create a new Document Pack and then use `ITPRun Result (_document_pack) OutputMode ("pack") …;` to create content.
- Activate a Document Pack with the indicated `Name` in the session that was previously copied using `CopyDocumentPack` or discarded using `CloseDocumentPack KeepTree (True).`
- Load a previously saved Document Pack from disk using the `LoadFrom` parameter. This file must have been created using the command `SaveDocumentPack` (see SaveDocumentPack).

CreatePath

The command `CreatePath` is an extended version of the command `CreateDirectory` that creates all directories in the given path.

`CreatePath` is implemented as a script component that is part of the built-in CCM Core script library. The script for `CreatePath` resides in: `<deploy root>\CCM\Documentation\5.2\Resources\Examples\Core Scripting`. For more information on how to use these examples, see Examples of script components.

Usage

```
CreatePath
      Path (<text>)
      Security (<text>);
```

Parameters

- `Path`: Required. The directory that CCM Core creates.
- `Security`: Optional. Directory used to set security attributes on the new directory.

CreateSession

CCM Core creates a new session and associates it with the active job. Every session has a unique storage area used to store data associated with that session. For more information on sessions, see CCM Core sessions.

The command `CreateSession` fails if the active job is already associated with a session.

Syntax

```
CreateSession
   Persistent(True or False);
```

Parameters

`Persistent`: Optional. Indicates whether the session should be automatically removed when the job ends. If this parameter is not specified, the session is not automatically removed.

The command `CreateSession` creates a directory structure in the session folder. If the session is persistent this structure is only removed when the session is closed by the command `CleanupSession` or when it is expired using the command `ExpireSessions`. Scripts can access this directory through the variable _sessiondir.

The command `CreateSession` generates a unique session identifier and puts this in the variable _sessionid. Subsequent jobs submitted with this session identifier are automatically associated with the session. The business application developer is responsible for returning the session identifier to the application and passing it back on subsequent calls. All calls with the same session identifier are serialized by CCM Core.

Delete

CCM Core deletes a specified file. The command `Delete` succeeds if the `File` does not exist.

Syntax

```
Delete
   File(<text>)
   TimeOut(<number>);
```

Parameters

- `File`: Required. The file that CCM Core deletes.
- `TimeOut`: Optional. The maximum amount of time in seconds that CCM Core waits if another process is locking the File. If this parameter is not specified, CCM Core waits for the file to be unlocked. If the value of this parameter is 0 and File is locked, CCM Core immediately reports an error.

DistributeDocumentPackToOutputManagement

The `DistributeDocumentPackToOutputManagement` command distributes a zipped Document Pack and an Import Request XML to the CCM Batch & Output Management Input Request folder, which is the destination folder.

The command retrieves the destination folder, which can be configured with the `OutputManagementHotfolder` parameter in the Contract Manager (for a description of the parameter, see the *Kofax Customer Communications Manager Getting Started Guide*). When the Output Management Hotfolder is not available, the command uses the destination folder entered for its optional `OutputManagementFolder` parameter (see the description of this parameter later in this section).

If you do not already have a zipped Document Pack file, you can use the `SaveDocumentPack` command (see SaveDocumentPack). The result file of this command is the zipped Document Pack with the name that can be specified in the `DocumentPackZip` parameter.

The command `DistributeDocumentPackToOutputManagement` requires a session. The `DistributionName` under which the zipped Document Pack and the Import Request XML are distributed is available as `get_sessionparameter` under the key `_outputmanagement_distribute_name`.

This command does not validate the zipped Document Pack or Import Request XML.

Parameters

- `DocumentPackZip`: The name of the zipped Document Pack to be distributed to CCM Batch & Output Management.
- `RequestXml`: An Import Request XML that conforms to CcmBomRequest.xsd. This XML must contain a valid Import Request to be picked up by CCM Batch & Output Management. For more information on the Import Request XML, see the *Batch & Output Management Getting Started Guide*.
- `OutputManagementFolder`: Optional. A destination folder to distribute to. The value of this parameter is only used if the Output Management Hotfolder destination cannot be retrieved from the Contract Manager. This may occur if the `OutputManagementHotfolder` parameter was not configured for a contract, or when this command is used in a script that is not executed through the Contract Manager.

Example

```
 Var Text DocumentPack = "c:\temp\documentpack.zip";
 Var Text ImportRequest = "c:\temp\importrequest.xml";

DistributeDocumentPackToOutputManagement
  DocumentPackZip(DocumentPack)
  RequestXml(ImportRequest);

Var Text DistributionName = get_sessionparameter(" _outputmanagement_distribute_name
 ");
```

DocToPDF

With the `DocToPDF` command you can directly generate PDF files from documents. CCM Core opens the `Src` document and saves this document to the file `Dest` in PDF format. For this conversion CCM Core uses either Microsoft Word and the Amyuni printer driver installed with the CCM Core software or the built-in Rendition technology.

If a Microsoft Word installation is present, CCM Core uses Microsoft Word by default. If no Microsoft Word installation is present, CCM Core uses Rendition by default. The conversion technology can be explicitly set through the global DocToPDF.Processor setting in the dp.ini file, or through the Processor(...) parameter on the `DocToPDF` command.

Microsoft Word

Using Microsoft Word in combination with the Amyuni printer driver to convert documents to PDF has the following advantages:

- Microsoft Word is used to render the document through the Amyuni printer driver. The PDF output is identical to other printed output.
- Both DOC and DOCX documents can be converted.
- The Amyuni conversion supports all configuration options described in this chapter.

The main disadvantage of this technology is the dependency on an installed copy of Microsoft Word on the server.

Rendition

Rendition is an alternative rendering engine included with CCM Core. This engine does not require a Microsoft Word installation on the CCM Core server. The Rendition engine can produce hyperlinks in the PDF document and can produce bookmarks based on the structure of the Word document.

The Rendition engine has the following limitations:

- Only DOCX documents can be converted. DOC documents are not supported.
- The converted PDF document can differ slightly from the document produced by Microsoft Word, and some DOCX features are not supported.
- Not all `DocToPDF` configuration options are supported.

Syntax

```
DocToPDF
   Src(<text>)
   Dest(<text>)
   TimeOut(<text>)
   EmbedFonts(True or False)
   EmbedFullFonts(True or False)
   EmbedStandardFonts(True or False)
   EmbedLicensedFonts(True or False)
   SubstitutionFont (<text>)
   WaterMarkText(<text>)
   WaterMarkFont(<text>)
   WaterMarkSize(<text>)
   WaterMarkColour(<number>)
   WaterMarkPosition(<text>)
```

```
WaterMarkAngle(<number>)
WaterMarkForeground(True or False)
WebOptimise(True or False)
ContentCompression(True or False)
JPEGCompression(<number>)
CCITTCompression(True or False)
AutomaticImageCompression(True or False)
Image256ColourReduction(True or False)
GreyScale(True or False)
ProducePDFA (True or False)
DPI (<number>)
ProducePDFX (True or False)
ColorProfile (<text>)
Processor("Word" or "Rendition")
Outline("", "bookmarks" or "headings")
```

Parameters

- `Src`: Required. The file that CCM Core converts to PDF format.
- `Dest`: Required. The destination of the converted file.
- `TimeOut`: Optional. The timeout for this command in seconds. If the word processor exceeds this timeout, the process is terminated and CCM Core reports a run-time error. If this parameter is omitted, the appropriate default timeout value for either batch or interactive jobs is used. A timeout value of 0 disables the timeout.
- `Processor`: Optional. Selects the conversion technology used to produce the PDF document. If this parameter is set to "Word", Microsoft Word and the Amyuni printer driver are used. If this parameter is set to "Rendition", the Rendition engine is used. This parameter defaults to "Word" when Microsoft Word is installed on the server, or "Rendition" if Microsoft Word is not installed. The default can be changed through the following setting in the dp.ini file:

  ```
  DocToPDF.Processor=<"Word" or "Rendition">
  ```

- `WaterMarkText`: Optional. Produces a text as a watermark on every page. When omitted, no watermark is included.

  Note The Rendition engine cannot produce watermarks. All watermark-related parameters are ignored. Watermarks are only available on 32-bit Microsoft Windows platforms if Microsoft Office is also a 32-bit version, and on 64-bit Microsoft Windows platforms if Microsoft Office is a 64-bit version. If a 32-bit Microsoft Office is installed on a 64-bit Microsoft Windows platform, the Watermark options have no effect.

- `WaterMarkFont`: Optional. The font for the watermark text. When omitted the CourierNew font is used.
- `WaterMarkSize`: Optional. The font size for the watermark text. This parameter can be expressed in inches ("<n>;inch") or centimeters ("<n>;cm"). When omitted a 0.5 inch ("0.5;inch") font size is used.
- `WaterMarkColour`: Optional. The color of the watermark in BGR (blue-green-red). This value must be expressed in the hexadecimal format 0xBBGGRR where BB represents the blue component of the color, GG the green component and RR the red component. When omitted, the watermark with the color gray (0x808080) is used.
- `WaterMarkPosition`: Optional. The position of the watermark relative to the upper left corner of the page. The position can be specified in inches ("<x>;<y>;inch") or centimeters ("<x>;<y>;cm"). When omitted, the watermark is anchored to the upper left corner of the document.
- `WaterMarkAngle`: Optional. The angle the watermark is rotated. When omitted, the watermark is not rotated.

- `WaterMarkForeground`: Optional. When set to True, the watermark is positioned over the content of the page. When omitted or set to False, the watermark is placed under the content.
- `EmbedFonts`: Optional. If this parameter is set to True, the conversion process embeds all non-standard PostScript fonts. This parameter is by default set to False. The default can be changed to True through the following setting in the dp.ini file:

  ```
  DocToPDF.EmbedFonts=Y
  ```

  Note The Rendition engine always embeds subsets of any non-standard PDF font. All parameters related to font embedding are ignored. Also, when using Microsoft Word, this parameter only has effect when the MultiLingual setting has been disabled. If the MultiLingual setting is enabled, fonts are always embedded.

- `EmbedFullFonts`: Optional. If this parameter is set to False, only a subset containing the characters actually used in the document is embedded. Otherwise full fonts are embedded. This parameter is by default set to False. The default can be changed to True through the following setting in the dp.ini file:

  ```
  DocToPDF.EmbedFullFonts=Y
  ```

  Note This parameter is ignored when the EmbedFonts setting is disabled.

- `EmbedStandardFonts`: Optional. If this parameter is set to True, standard PDF fonts are embedded in the PDF file. This parameter is by default set to False. The default can be changed to True through the following setting in the dp.ini file:

  ```
  DocToPDF.EmbedStandardFonts=Y
  ```

  This parameter only has effect when the MultiLingual setting has been disabled. If the MultiLingual setting is enabled, standard PDF fonts are always embedded.
- `SubstitutionFont`: Optional. The font to use when a document has a font that is not present on the system. The setting only has effect if the Rendition engine is used. Possible values are:
  - `*default` Rendition uses Arial Unicode MS if it is installed on the system. If not, it uses the font specified as the default font for the document (usually Times New Roman).
  - `*none` Rendition does not substitute fonts but shows an error message when a font is not present.
  - Name of the substitution font. Rendition uses this font as a substitution. If it is not present, an error message is shown.

  If this parameter is omitted, the default value of the Rendition.SubstitutionFont global setting is used. The default can be changed through the following setting in the dp.ini file:

  ```
  Rendition.SubstitutionFont=Arial Unicode MS
  ```
- `EmbedLicensedFonts`: Optional. If this parameter is set to True, licensed fonts are embedded in the PDF file. This parameter is by default set to False. The default can be changed to True through the following setting in the dp.ini file:

  ```
  DocToPDF.EmbedLicensedFonts=Y
  ```
- `WebOptimise`: Optional. If this parameter is set to True, a Web Optimized (linearized) PDF is produced. This parameter is by default set to False. The default can be changed to True through the following setting in the dp.ini file:

  ```
  DocToPDF.WebOptimise=Y
  ```

  The Rendition engine does not produce linearized PDF files. This parameter is ignored.

Note When this option is set to True, Microsoft Word ignores the ProducePDFA parameter. The resulting PDF file is not PDF/A-1b compliant.

- `ContentCompression`: Optional. If this parameter is set to False, the content of the PDF file is not compressed. This parameter is by default set to True. The default can be changed to False through the following setting in the dp.ini file:

  ```
  DocToPDF.ContentCompression=N
  ```

  The Rendition engine always compresses the PDF content. This parameter is ignored.
- `AutomaticImageCompression`: Optional. If this parameter is set to False, images are embedded using a combination of run-length encoding and compression. If this parameter is set to False, each image is embedded using a compression method selected based on the characteristics of the image. This parameter is by default set to True. The Rendition engine always compresses images. This parameter is ignored.
- `JPEGCompression`: Optional. Specifies the compression rate for 24-bit images. Values indicate trade-off between size and quality, and must be in the range from 0 (no compression) to 9 (high compression, low quality). This parameter is by default set to 0 (no compression). The Rendition engine ignores this setting. This setting is ignored when AutomaticImageCompression is True. We recommend you to use a single value, or only switch between the values 0, 1, 3 or 8. Switching to or from one of the other values will cause significant overhead.
- `CCITTCompression`: Optional. If this setting is set to True, black and white images are compressed using CCITT compression. This parameter is by default set to False. The Rendition engine does not use CCITT compression. This parameter is ignored.
- `Image256ColourReduction`: Optional. If this setting is set to True, all images are reduced to a 256-color palette. This can cause a significant loss of quality in 24-bit images. This parameter is by default set to False. This setting only has effect when Automatic image compression has been turned off.
- `GreyScale`: Optional. If this parameter is set to True, all color in the document is dithered to grayscale. This parameter is by default set to False. The Rendition engine ignores this setting.
- `ProducePDFA`: Optional. If this setting is set to True, the PDF document is PDF/A-1b compliant and suitable for archival as defined in ISO 19005 Part-1 Level B. This parameter is by default set to False. The default can be changed to True through the following setting in the dp.ini file:

  ```
  DocToPDF.ExportPDFA=Y
  ```

  The PDF/A-1b specification explicitly excludes the use of transparency. Microsoft Word silently ignores the transparency which could result in unexpected output. The Rendition engine reports an error when the document contains transparent objects. The Rendition engine does not support multiline borders for PDF/A-1b. Such borders are rendered as a single solid border. Microsoft Word ignores this setting when the WebOptimise parameter is set to True.
- `DPI`: Optional. Overrides the default resolution used for positioning text and downsampling images. When using Microsoft Word, the supported resolutions are limited to 72, 150, 300, 600 and 1200 DPI. This parameter is by default set to 300 DPI. The default can be changed to another resolution through the following setting in the dp.ini file:

  ```
  DocToPDF.DPI=<resolution>
  ```

  Microsoft Word uses this setting for positioning text and downsampling images. The Rendition engine uses this setting when converting vector graphics to bitmaps and for rendering EMF/WMF graphics. Images included in the PDF file are not downsampled.

- `ProducePDFX`: Optional. If set to True, the result document is PDF/X-3 compliant. This parameter is by default set to False. The default can be changed to True through the following setting in the dp.ini file:

  ```
  DocToPDF.ExportPDFX=Y
  ```

  The Rendition engine does not produce PDF/X-3 compliant documents. This parameter is ignored. Enabling the PDF/X support also enables Image Compression.
- `ColorProfile`: Optional. The color profile file used for PDF/X creation. The color profile can be either the name of the profile with the extension, such as AmyuniCMYK.icc, if the color profile is already installed on the system, or the full path to where the profile is located. There is no default value for this setting. A default color profile can be set through the following setting in the dp.ini file:

  ```
  DocToPDF.ColorProfile=<colorprofile>
  ```

  The Rendition engine ignores this setting. For PDF/A the sRGB color profile is used. A valid color profile is required to produce PDF/X documents. Color profiles are not supported for non-PDF/X documents.
- `Outline`: Optional. Produce bookmarks in the PDF document representing the logical structure of the document. If this parameter is omitted or empty, no bookmarks are produced. If this parameter is set to "headings", heading styles are used to generate a nested bookmark structure. If this parameter is set to "bookmarks", bookmarks in the document are used to generate the bookmarks as a flat list. The default can be changed through the following setting in the dp.ini file:

  ```
  Rendition.Outline=<default>
  ```

  The Amyuni engine cannot produce outlines and ignores this parameter.

  This parameter is introduced in CCM Core version 5.1.1.

Microsoft Word 2003 compatibility mode

Converting Microsoft Word documents in the DOCX format can cost a significant amount of time when these documents contain graphics. Performance can be improved by forcing Microsoft Word 2010 and later versions to use the Microsoft Word 2003 Compatibility Mode. This mode uses an older version of the graphics engine built in Microsoft Word which is also used for DOC documents.

CCM Core can be configured to force this compatibility mode by adding the following setting to the [Configuration] section of the dp.ini file.

```
Word2003CompatibleMode=Y
```

Forcing the compatibility mode can have an effect on the quality of the graphics and other content of the document.

This setting has no effect on Microsoft Word 2007 or older versions.

Note The Rendition engine ignores this setting.

This setting was introduced in CCM Core version 4.2.3.

Global settings

A number of settings in the [Configuration] section of the dp.ini file of the CCM Core installation change the behaviour of the `DocToPDF` command. These settings are specific for the selected conversion technology and are ignored when not applicable.

Settings that affect default parameter values are described in DocToPDF.

Microsoft Word

`DocToPDF.PaperSize`

Sets the default paper size. Normally, this setting does not need to be specified, as the paper size is derived from the source document.

```
DocToPDF.PaperSize=<"A4", "A3", "Letter", or "Legal">
```

If not set, the paper size defaults to A4.

`DocToPDF.Orientation`

Sets the default paper orientation. Possible values are Landscape and Portrait.

```
DocToPDF.Orientation=<"Portrait" or "Landscape">
```

If not set, the orientation defaults to Portrait.

`DocToPDF.MultiLingual`

Set to N to turn off multilanguage support. Multilanguage support is needed to support non-western and special character sets, and gives the most portable documents in practice. This setting should be turned on to avoid problems with missing characters in the output. This setting is often required when the output contains currency symbols such as the euro sign (€). Turning off multilingual support may result in smaller files.

`DocToPDF.MultiLingual=<Y or N>`

If not set, it defaults to Y. Note that if multilingual support is on, most font embedding parameters have no effect. In that case, all fonts, including the standard fonts but with the exception of licensed fonts, are embedded as a subset.

`DocToPDF.HorizontalMargin`

Sets the horizontal printer margins (in mm). The printer margin defines the area that contains graphical data. The area outside the margin stays blank. Normally, this setting does not need to be specified, as the default allows the entire page to be used.

`DocToPDF.HorizontalMargin=<number>`

If not set, it defaults to zero.

`DocToPDF.VerticalMargin`

Sets the vertical printer margins (in mm). The printer margin defines the area that contains graphical data. The area outside the margin stays blank. Normally, this setting does not need to be specified, as the default allows the entire page to be used.

`DocToPDF.VerticalMargin=<number>`

If not set, it defaults to zero.

`DocToPDF.SimPostscript`

Set to Y to enable Postscript simulation, or N to disable. This option changes how the Amyuni PDF printer driver presents itself to the word processor application. This option is needed when using transparent backgrounds or watermarks in Microsoft Word.

```
DocToPDF.SimPostscript=<Y or N>
```

Rendition

```
Rendition.IgnoreUnsupportedContent
```

Set to N to have `DocToPDF` reject documents that contain content known to be unsupported. By default, the Rendition engine silently ignores unsupported content.

```
Rendition.IgnoreUnsupportedContent=<Y or N>
```

If not set, it defaults to Y.

Clearing this flag does not guarantee that the produced PDF documents are the same as the output produced by Microsoft Word. It flags a number of known issues.

```
Rendition.StrictParameterCheck
```

Set to Y to have the `DocToPDF` command report an error when it encounters an unsupported parameter or setting if `Processor` is set to `Rendition`.

```
Rendition.StrictParameterCheck=<Y or N>
```

If not set, it defaults to N.

The following parameters are ignored even when this setting is enabled:

- `ContentCompression`
- `AutoImageCompression`
- `CCITTCompression`

Known issues when using Rendition

The Rendition engine supports the Word features listed in PDC Supported Word Features version 6.1.

In addition, there are some known restrictions when converting DOCX documents to PDF. The following features are currently not supported:

- Multiline borders when generating PDF/A. For non PDF/A documents, multiline borders are supported.
- Pattern fills
- Linear gradient fills with an angle that is not a multiple of 45 degrees
- Radial gradients fills that do not have their center in the middle of the shaded area
- Graphical objects that are flipped (mirrored)
- Shapes with 3D effects

Configuration and performance when using Microsoft Word

The CCM Core services must be running with local or domain credentials for the `DocToPDF` command to function correctly. Running the services with Local System credentials is not supported.

Some of the configuration parameters are applied to the Amyuni printer driver when it is loaded during startup. Some non-default parameter values can require a reload of the printer driver which has a performance penalty.

The Rendition engine and associated configuration are introduced in CCM Core version 5.1.

Support for bookmarks and hyperlinks in the Rendition engine is introduced in CCM Core version 5.1.1.

ExpireSessions

CCM Core scans the active sessions and removes all sessions that meet the specified expiration criteria. For more information, see CCM Core sessions.

Syntax

```
ExpireSessions
   Age(<number>)
   LastAccess(<number>);
```

Parameters

- `Age`: Optional. Expires session created more than <number> minutes ago. If this parameter is omitted or set to 0, no expiration based on creation time is performed.
- `LastAccess`: Optional. Expires session last used more than <number> minutes ago. If this parameter is omitted or set to 0, no expiration based on last usage is performed.

The command `ExpireSessions` fails if both the parameters `Age` and `LastUsed` are either omitted or have the value 0. You can specify values for both parameters.

After the command `ExpireSessions` completes successfully, the variable _expired_sessions contains a comma-separated list of the expired sessions This variable is empty if no sessions were expired.

ExportDocToPDF

The `ExportDocToPDF` command uses the Microsoft Word export filter to produce a PDF file directly from a Microsoft Word document.

This command is only available for Microsoft Word documents and requires Microsoft Office 2007 or higher. This command might also require installation of the optional PDF/XPS add-in if PDF support is not included in the Microsoft Office installation.

The `ExportDocToPDF` command provides a number of optional parameters, which affect the way in which the PDF document is generated. If no optional parameters are provided, CCM Core uses sensible defaults.

You can set a global default for some of the optional parameters. These global defaults can be set in the [Configuration] section of the dp.ini file.

Syntax

```
ExportDocToPDF
```

```
Src(<text>)
Dest(<text>)
OptimiseForPrint(True or False)
IncludeProperties(True or False)
ExportBookmarks(<number>)
BitmapFonts(True or False)
ProducePDFA(True or False)
AccessibilityInfo(True or False)
TimeOut(<number>);
```

Parameters

- `Src`: Required. The Microsoft Word document that CCM Core converts to PDF.
- `Dest`: Required. The resulting PDF document.
- `OptimiseForPrint`: Optional. Indicates if the resulting PDF document should be optimized for printing (True) or for screen reading (False). This parameter defaults to False when omitted. You can change the default value using the following setting in the dp.ini file.
- `ExportDocToPDF.OptimiseForPrint= (Y or N)`
- `IncludeProperties`: Optional. Indicates whether some document properties, such as author, title, and so on, from the Microsoft Word document should be copied into the PDF document. This parameter defaults to False when omitted.
- `ExportBookmarks`: Optional. Indicates whether a bookmark table should be added to the PDF document. Supported values are:
  - 0 No bookmarks are generated
  - 1 Bookmarks are generated based on the Heading styles in the Word document
  - 2 Bookmarks are generated based on bookmarks in the Word document
- This parameter defaults to 0 when omitted.
- `BitmapFonts`: Optional. If font licenses do not permit a font to be embedded in the PDF document, this setting indicates how the font is handled. True: a bitmap of the text is included in the PDF document. False: the font is referred but not included. Select an appropriate (substitute) font in the printer settings. This parameter defaults to True when omitted. You can change the default value using the following setting in the dp.ini file:

  ```
  ExportDocToPDF.FontBitmaps=(Y or N)
  ```

- `ProducePDFA`: Optional. Indicates whether the PDF document is limited to the PDF/A (ISO 19005-1) subset. Such documents are more self-contained but could be larger or show more visual artifacts. This parameter defaults to False when omitted but the default value can be changed using the following setting in the dp.ini file:

  ```
  ExportDocToPDF.ExportPDFA=(Y or N)
  ```

- `AccessibilityInfo`: Optional. Indicates whether structure information is embedded in the PDF document to facilitate screen readers. This parameter defaults to True when omitted.
- `TimeOut`: Optional. The timeout for this command in seconds. If the word processor exceeds this timeout, the process is terminated and CCM Core reports a run-time error. If this parameter is omitted, the appropriate default timeout value for either batch or interactive jobs is used. A timeout value of 0 disables the timeout. This feature is currently only supported for Microsoft Word.

Global defaults

You can set global defaults for the `ExportDocToPDF` command in the [Configuration] section of the dp.ini file.

```
ExportDocToPDF.OptimiseForPrint
```

Set to Y to optimize the PDF document for printing. This improves the rendering of the PDF file but generates larger PDF files. Set to N to optimize the PDF document for viewing on a screen.

```
ExportDocToPDF.OptimiseForPrint=(Y or N)
```

If omitted, this setting defaults to N.

```
ExportDocToPDF.FontBitmaps
```

Determine the default handling of fonts with a license that forbids inclusion into the PDF document. Set to Y to include these fonts as bitmaps. Set to N to include a reference to the font and leave it to the viewer to select an appropriate (substitute) font.

```
 ExportDocToPDF.FontBitmaps=(Y or N)
```

If omitted, this setting defaults to Y.

```
ExportDocToPDF.ExportPDFA
```

Set to Y to limit the PDF document to the PDF/A (ISO 19005-1) subset. Such documents are more self-contained but could be larger or show more visual artifacts.

```
 ExportDocToPDF.ExportPDFA=(Y or N)
```

If omitted, this setting defaults to N.

ExportDocToXPS

The `ExportDocToXPS` command uses the Microsoft Word export filter to produce an XPS file directly from a Microsoft Word document.

This command is only available for Microsoft Word documents and requires Microsoft Office 2007 or higher. This command might also require installation of the optional PDF/XPS add-in if XPS support is not included in the Microsoft Office installation.

The `ExportDocToXPS` command provides a number of optional parameters affecting the way in which the XPS document is generated. If no optional parameters are provided, CCM Core uses sensible defaults.

You can set a global default for some of the optional parameters in the [Configuration] section of the dp.ini file.

Syntax

```
ExportDocToXPS
   Src(<text>)
   Dest(<text>)
   OptimiseForPrint(True or False)
   IncludeProperties(True or False)
   ExportBookmarks(<number>)
   BitmapFonts(True or False)
   KeepIRM(True or False)
   AccessibilityInfo(True or False)
   TimeOut(<number>);
```

Parameters

- `Src`: Required. The Microsoft Word document that CCM Core converts to XPS.
- `Dest`: Required. The resulting XPS document.
- `OptimiseForPrint`: Optional. Indicates if the resulting XPS document should be optimized for printing (True) or for screen reading (False). This parameter defaults to False when omitted, but the default value can be changed using the following setting in the dp.ini file:

  ```
  ExportDocToXPS.OptimiseForPrint=(Y or N)
  ```
- `IncludeProperties`: Optional. Indicates whether some document properties, such as author, title, and so on, from the Microsoft Word document should be copied into the XPS document. This parameter defaults to False when omitted.
- `ExportBookmarks`: Optional. Indicates whether a bookmark table should be added to the XPS document.

Supported values are:

- 0 No bookmarks are generated
- 1 Bookmarks are generated based on the Heading styles in the Word document
- 2 Bookmarks are generated based on bookmarks in the Word document

This parameter defaults to 0 when omitted.

`BitmapFonts`: Optional. If font licenses do not permit a font to be embedded in the XPS document, this setting indicates how the font is handled. True: a bitmap of the text is included in the XPS document. False: the font is referred but not included. Select an appropriate (substitute) font in the printer settings.

This parameter defaults to True when omitted. You can change the default value using the following setting in the dp.ini file.

```
ExportDocToXPS.FontBitmaps=(Y or N)
```

`KeepIRM`: Optional. Indicates whether IRM information should be copied from the Microsoft Word document into the XPS document. This parameter defaults to True when omitted.

`AccessibilityInfo`: Optional. Indicates whether structure information is embedded in the XPS document to facilitate screen readers.

This parameter defaults to True when omitted.

`TimeOut`: Optional. The timeout for this command in seconds. If the word processor exceeds this timeout, the process is terminated and CCM Core reports a run-time error. If this parameter is omitted, the appropriate default timeout value for either batch or interactive jobs is used. A timeout value of 0 disables the timeout. This feature is currently only supported for Microsoft Word.

Global defaults

You can set global defaults for the `ExportDocToXPS` command in the [Configuration] section of the dp.ini file.

```
   ExportDocToXPS.OptimiseForPrint
```

Set to Y to optimize the XPS document for printing. This improves the rendering of the XPS file but generates larger XPS files. Set to N to optimize the XPS document for viewing on a screen.

```
   ExportDocToXPS.OptimiseForPrint=(Y or N)
```

If omitted, this setting defaults to N.

```
   ExportDocToXPS.FontBitmaps
```

Determine the default handling of fonts with a license that forbids inclusion into the XPS document. Set to Y to include these fonts as bitmaps. Set to N to include a reference to the font.

```
   ExportDocToXPS.FontBitmaps=(Y or N)
```

If omitted, this setting defaults to Y.

FTP

CCM Core uploads a file to any host that understands the ARPANET File Transfer Protocol (FTP).

The host must run an FTPD daemon. This daemon must support the minimum requirements as described in RFC 959. All files are uploaded in binary (image) mode. Most hosts support anonymous uploads for the user "ftp" with any valid email address as its password.

CCM Core does not log the transmitted password.

Syntax

```
FTP
   File(<text>)
   Host(<text>)
   Port(<text>)
   User(<text>)
   Password(<text>)
   Directory(<text>)
   Passive(True or False);
```

Parameters

- `File`: Required. The file that CCM Core uploads.
- `Host`: Required. The host where CCM Core uploads the file to.
- `Port`: Optional. The number or symbolic name of the port that CCM Core connects to on the host. If this parameter is not specified, CCM Core uses the standard FTP port (port 21).
- `User`: Optional. The User ID that CCM Core uses when connecting to the host. If this parameter is not specified, CCM Core connects as user "ftp" to initiate an anonymous FTP session.
- `Password`: Optional. The password that CCM Core uses when connecting to the host. If this parameter is not set, CCM Core sends the user name to the host. Note that this password must be supplied in plain text (not encrypted).
- `Directory`: Optional. Directory on the host where CCM Core uploads the file. If this parameter is not specified, CCM Core uploads the file to the default directory of the user with which it has connected to the host.
- `Passive`: Optional. Indicates whether CCM Core should use Passive FTP to upload the file to the FTP server. This option should be False unless there is a firewall or NAT server between CCM Core and the FTP server.

InsertDocumentPack

Use the command `InsertDocumentPack` to add or replace an alternative format document into the active Document Pack.

The command `InsertDocumentPack` fails if there is no active Document Pack in the session.

Syntax

```
InsertDocumentPack
      Document (<text>)
      Type (<text>)
      Slot (<text>)
      Channel (<text>)
      Flags (<text>);
```

Parameters

- `Document`: Required. Specifies the file that is inserted or replaced in the Document Pack. This file must be located under the Data subdirectory of the directory indicated by the _document_pack constant.
- `Type`: Required. Specifies the file type for the document.
- `Slot`: Required. Indicates the slot in the Document Pack that is inserted or replaced. It is only possible to insert files into an existing slot.
- `Channel`: Optional. Indicates that the file is to be inserted into a specific channel exception for the slot. This channel exception must already be present in the Document Pack. If this parameter is omitted or empty, the file is inserted in the default channel.
- `Flags`: Optional. Specifies the flags for the alternative.

The command `InsertDocumentPack` uses the `Slot` and `Channel` values to identify the document in the Document Pack. It is not possible to introduce new slots or channel exceptions.

If the document is found alternatives are matched based on the `Type` and `Flags` values. If there is already an alternative with these attributes, it is replaced. Otherwise, a new alternative is added to the slot.

The command `InsertDocumentPack` must be used to change the manifest when an entry should refer to a new document. If the document is changed locally, such as when applying a macro or securing a PDF file, it is not necessary to update the manifest.

IterateDocumentPack

Use the command `IterateDocumentPack` to iterate through documents in the active Document Pack. For each document, a script is called to perform actions.

The command `IterateDocumentPack` fails if there is no active Document Pack in the session.

Syntax

```
IterateDocumentPack
```

```
    Script (<text>)
    Context (<text>)
    Status (<text>)
    Slot (<text>)
    Channel (<text>)
    Type (<text>)
    DelayErrors (True or False);
```

Parameters

- `Script`: Required. The script that is called for each document in the Document Pack.
- `Context`: Required. This value is directly passed to the `Context` parameter of the script.
- `Status`: Optional. Filter. If this parameter is omitted, all formats are processed.
- `Slot`: Optional. Filter. If this parameter is omitted, all slots are processed.
- `Channel`: Optional. Filter. If this parameter is omitted, all channels are processed. If this parameter is empty, only the default channels are processed. Otherwise, only matching channel exceptions are processed.
- `Type`: Optional. Filter.
- `DelayErrors`: Optional. Indicates whether a failure in the iterator script should cause the iteration to terminate immediately or keep processing documents. If this parameter is omitted, processing stops immediately if an error occurs.

The command `IterateDocumentPack` processes all records in the manifest of the active Document Pack as it was at the time the command was called. The processing script is allowed to modify the content of the Document Pack, but this does not affect the iteration. The processing script is allowed to use the command `IterateDocumentPack` recursively.

The parameters `Status`, `Type`, `Slot`, and `Channel` are used to select documents for processing. Documents that do not match the parameter are ignored during the iteration.

The `Status` parameter can have one of the following values:

- `"T"` to select the original output of the template.
- `"I"` to select import documents.
- `"A"` to select alternative formats of the output. There can be multiple alternative formats associated with a template, resulting in multiple calls to the script.

The command `IterateDocumentPack` requires a script with the name indicated by the `Script` parameter. This script can define the following parameters:

- `Document`. The fully qualified path to the document in the session directory or "`*none`" if there is explicitly no document produced for the channel.
- `Template`. The CCM template that produced the document.
- `Type`. The output type of the document.
- `Slot`. The slot in the Document Pack.
- `Channel`. The channel. This parameter is empty for the default channel.
- `Status`. Type of the document.
- `ClosedLoopIdentifier`. The value of the Closed Loop Identifier for this entry.
- `Flags`.
- `Metadata`. The fully qualified path to the Metadata XML file. This parameter is empty if there is no metadata written.

- `DataBackbone`. The fully qualified path to the Data Backbone XML that was produced when the template was processed. This parameter is empty for static templates and import documents.
- `Context`. The value of the `Context` parameter on the command.

All these parameters have type Text. Parameters can be omitted if the value is not used in the script.

Example

This example lists all slots in the Document Pack.

```
IterateDocumentPack
      Script (ListSlots)
      Channel (“”)       # Filter default channels
      Context (“TI”);    # Passed to the iteration script.
                         # Select Templates and Imports, ignore
                         # converted alternatives.
```

The ListSlots.dss script.

```
Parameter Text Slot;
Parameter Text Status;
Parameter Text Context;
/* Ignore other parameters. */

# List Slots whose Status is in the Context list.
If index (Context, Status) > 0 Then
 Progress Message (Slot);
Fi;
```

If any call to the iterator script ends in an error, the command `IterateDocumentPack` immediately fails. You can set the `DelayErrors` parameter to True to continue processing and delay the failure until after all alternatives have been processed.

When errors are delayed, the value of the _message global constant is undefined.

ITPError

The `ITPError` command to handle error situations in commands is deprecated.

This command does the following:

- It returns the original error, the internal ITPLOG messages, and the internal ITPLOGDM messages as progress messages.
- It writes an itperror.log file next to the log.
- It re-throws the error, extended with the internal ITPLOG and ITPLOGDM messages.

`ITPError` saves the following files to the folder indicated at the parameter ErrorDir:

- dm.ini
- ITP configuration file (itp.cfg) that was active at the time of the error including all settings that were generated automatically.
- postinc.doc: this is the result document in which the post-include statements have not been processed yet. This file is useful to debug post include paths.

- itperror.log. This file contains all other relevant information for ITP jobs that have failed. In particular, it contains the information from ITP log and the ITP Data Manager log files. The information in the ErrorDir is appended so that it contains history for this particular ErrorDir.

Usage

To install the ITPError script as an error handler, use the following commands.

```
ITPErrorReset; /* Only the first time */
OnError Script(ITPError)
   Model("Model")
   ErrorDir ("Path\folder")
;
```

A closing semicolon is required.

Note The usage of the command `ITPErrorReset` (see ITPErrorReset) before `ITPError` is installed as an error handler. This command is required to ensure that only relevant information is included in the error report. It should be called once, before the first time that the `ITPError` handler is installed. When the `ITPError` handler is reinstalled, such as after disabling error handling using OnError Script(*), the command `ITPErrorReset` should not be re-issued.

Parameters

- `Model`: Required. Name of the template used in the CCM process of which the ITPError is the error handler.
- `ErrorDir`: Optional. The name of the folder used to save the files to. This folder is specified relative to the Log folder of a Document Processor. By default, the name of the folder is set to the job id with which the service is called.

ITPErrorReset

The `ITPErrorReset` command is deprecated.

You can only use this command in combination with the command `ITPError` (see ITPError).

ITPRun

You can use the `ITPRun` command to run a template. This command replaces the ITP command, which is deprecated.

Usage

```
ITPRun
      Model (<text>)
      Result (<text>)
```

The first two arguments are the name of the template and the location of the result. They are both mandatory. The remaining arguments are all optional.

A template can expect Keys and Extras in a certain order. They must be provided as a list of values, separated by semicolons.

```
Keys(<text>)
Extras(<text>)
DisablePostIncludes (True or False)
Environment (<text>)
```

The `MetaData` parameter specifies the name of an XML file to store template run metadata after the template completed.

```
MetaData (<text>)
```

The following parameters specify a user name and password for retrieving data in the template run.

```
DBUserID (<text>)
DBPassword (<text>)
```

The following section is AS/400 only and optional. See also AS/400 Connection parameters later in this section.

```
PreCMD (<text>)
OnSuccessCMD (<text>)
OnFailureCMD (<text>)
PostCMD (<text>)
```

You can specify the data XML file with which the Data Backbone is filled.

```
DBB_XMLInput (<text>)
DBB_XMLOutput (<text>)
BatchMode (True or False)
```

With `OutputMode` the formatting of the result document can be specified.

```
OutputMode (<text>)
```

Pre-flight checks can be disabled with the following setting.

```
SuppressInteractiveCheck (True or False)
```

Integration used with Kofax TotalAgility and other applications.

```
ClosedLoopIdentifier (<text>);
```

Parameters

- `Model`: Required, string. The rep:/ URI or Letter Book URL to be executed. For more details on the rep:/ URI, see the chapter "Document composition" in the *Kofax Customer Communications Manager Core Developer's Guide*.
- `Result`: Required, string. Path, name, and extension of the result document. The format of the result document is the same as the format of the template script used to create the template. If the template script is a Microsoft Word document (.docx), the result document is a Microsoft Word (.docx) document. The format of the result document can be overridden by the `OutputMode` parameter.
- `Keys`: Optional, string. Keys are used to specify parameters for the data retrieval of a template. A template can use data retrieval statements to retrieve data from external sources. Data retrieval statements are parameterized with keys used to identify which data should be retrieved, such as the number of the intended recipient. Keys are passed as a string of values separated by semicolons. Each value specifies a single parameter required by a data retrieval statement in the template.

Note Data retrieval parameters explicitly specified with the PAR keyword in the Template scripting language are not taken from the parameter Keys. Values from the parameter Keys are only used when a data retrieval statement in the template does not explicitly specify a parameter itself.

The sequence of keys passed in the parameter Keys must be in the order in which they are expected in the template. You can specify an empty parameter by following a semicolon with another semicolon. This ensures that the empty parameter counts in the sequence.

Example

```
ITPRun
      ...
      Keys ("value1;;value3;value4");
```

The meaning of a Keys entry can differ depending on the data connection being used. For database connections, a Keys entry typically provides a database primary key value, such as an invoice number used to retrieve records with a database query. For the XML File connection, a Keys entry specifies the full path to an XML file.

For the XML File connection, you can only specify XML files through the parameter Keys, not as parameters to the data retrieval statements in the template. For each data retrieval statement that uses a main entry from an XML File DID Module, the next entry in the parameter Keys determines the XML file to be used.

The script `ITPRun` does not support the use of parameter files.

`Extras`: Optional, string. Extra parameters are used to pass additional information to variables with the EXTRA keyword in the template. These parameters are usually calculated by the script or derived from the request parameters. Extra parameters are specified as a string of values separated by semicolons. The sequence must be in the same order as the variables in the template.

You can specify an empty parameter by following a semicolon with another semicolon. This ensures that the empty parameter counts in the sequence.

Example

```
ITPRun
      ...
      Extras ("value1;value2;;");
```

`DisablePostIncludes`: Optional, boolean. If set to True, CCM does not perform a post-include after the template has been run. Post-includes in CCM Core are performed lazily by default (ITPLAZYPOSTINC=Y). If set to False (or left empty), `ITPRun` uses the ITPPOSTINC setting in the itp.cfg to determine if post-includes have to be processed.

The default is False.

`Environment`: Optional, string. Templates are always run in an environment that can be indicated with this parameter. If no environment parameter is given, the default environment is used. If an environment is passed that does not exist, an error is given.

Note This parameter is ignored when running Document Pack Templates.

`MetaData`: Optional, string. The name of an XML file in which metadata on the template run is written. An existing metadata file may also be supplied to be extended by the template. If this parameter is not given, no such file is generated.

Note This parameter is ignored when running Document Pack Templates and with `OutputMode` set to "pack". For more information, see the section "Template run XML metadata" in the *Kofax Customer Communications Manager Core Developer's Guide*.

`DBUserID`: Optional, string. User account name for retrieving data from a database. This parameter allows a database user account to be specified per template run.

`DBPassword`: Optional, string. Password belonging to the account name specified in DBUserID.

`DBB_XMLInput`: Optional, string. The name of a data XML file used to fill the Data Backbone of the template. The data XML must match the XSD of the Data Backbone. In batch mode, this XSD is extended with wrappers to enable more than one Data Backbone data sets to be specified.

`DBB_XMLOutput`: Optional, string. The name of a file where the XML with data of the Data Backbone of the template should be stored after the template run has completed.

The file name passed to DBB_XMLInput and DBB_XMLOutput should be a valid path/file specification on the computer running CCM Core. It can be preceded by "session:", in which case the file is located in the session directory. The DBB parameters allow you to create templates that do not need a DID, as data is loaded directly into the Data Backbone from the XML.

`BatchMode`: Optional, boolean. Enable batch mode for CCM Interactive templates.

If batch mode is enabled, CCM Core expects one or more Data Backbone data sets in the input XML file. The model automatically fills the Data Backbone with one set at a time and runs the template for that set. The result document is a concatenated set of all output documents.

The default is False. Enabling this option requires that an XML file is passed using the DBB_XMLInput parameter.

`OutputMode`: Optional, string. Overrides the output format for the result document.

This parameter determines the type of the result document produced. Currently supported formats are:

- `native` produces a document in the same format as the template.
- `utf8` produces text in UTF-8 encoding.
- `utf16` produces text in UTF-16 encoding.
- `aiadocxml` produces an XML file representing the structure of the result document. This structure is based on the Content Wizard(s) used in the document.
- `pack` produces a Document Pack. If the template is a Document Template, it is treated as a Document Pack Template. The produced Document Pack will have a single slot containing the result of that template.

The default is `native`.

A template can query the `OutputMode` parameter by calling the function runmodel_setting("OutputMode").

`SuppressInteractiveCheck`: Optional, boolean. If set to True, CCM performs a pre-flight check to verify that a (potentially) interactive Master Template can be run in a non-interactive mode. The developer is responsible for detecting this and suppressing any interactivity. If the Master Template tries to become interactive, an error is reported. If set to False (or left empty), CCM prohibits Master Templates that contain interactive statements from running in a non-interactive mode.

`ClosedLoopIdentifier`: Optional, string.

Use the Closed Loop Identifier to pass an identifier to the template. The identifier will be included in the Metadata and used in the resulting Document Pack to identify each result document. Templates can override the identifier by changing the _Document.ClosedLoopIdentifier field.

This option is used in KTA integration. Use is also supported for other applications.

AS/400 exit points

The following commands allow the script to execute AS/400 shell commands to modify the context of the AS/400 HDM job.

`PreCMD`: Optional, string. The command `PreCMD` is executed after the library list is set.

`OnSuccessCMD`: Optional, string. The command `OnSuccessCMD` is executed if the template is completed successfully.

`OnFailureCMD`: Optional, string. The command `OnFailureCMD` is executed if the template failed.

`PostCMD`: Optional, string. The PostCMD command is executed at the end of the run (after `OnSuccessCMD` or `OnFailureCMD`).

Error handling

A template can use the keywords ERROR and WARNING to report errors to CCM Core. These errors are identified with the USR1000 and USR1001 labels.

Availability

As of CCM Core 4.2.0, references to templates on the file system are no longer supported.

The BatchMode and OutputMode parameters are introduced in CCM Core version 4.2.3.

The CCMOnDemandMode, CCMInterfaceType, and Alternative parameters are introduced in CCM Core version 4.4.

LogEvent

CCM Core sends a message to the NT Event Logging Service.

See the documentation of Microsoft Windows NT for an explanation of Source Identifiers, Event Types, and Event IDs. The default behavior of CCM Core is to write an error event to the event log with the text "Error: %1," where %1 is substituted by the text of the parameter Record.

The following Event IDs are available through the ITPSERVERMSG.DLL:

Event ID	(Hexadecimal)	Text
3221225672	0xC00000C8	Error: %1.
1073742025	0x400000C9	Warning: %1.
1073742026	0x400000CA	Progress: %1.
1073742027	0x400000CB	Info: %1.

The ITPSERVERMSG.DLL must be installed on the host to have it show the correct text in the Event Viewer.

Syntax

```
LogEvent
   Record(<text>)
   Host(<text>)
   EventType(<number>)
   EventID(<number>)
   Source(<text>);
```

Parameters

- `Record`: Required. The record (message text) sent to the Event Log.
- `Host`: Optional. The remote host where the event is logged. To see the correct text in the Event Viewer, ensure that ITPSERVERMSG.DLL is installed on the host. If this parameter is not specified, CCM Core logs the record on the local machine; ITPSERVERMSG.DLL is automatically installed on the host that CCM Core is installed on.
- `EventType`: Optional. The type of the event that CCM Core logs. If this parameter is not specified, CCM Core logs the record as type 1 (error).
- `EventID`: Optional. The ID of the event that CCM Core logs. If this parameter is not specified, CCM Core logs the record with an ID of 3221225672 or 0xC00000C8 (hexadecimal). See the table earlier in this section.
- `Source`: Optional. The source identifier of the event that CCM Core logs. If this parameter is not specified, CCM Core uses the source identifier "ITPEvent". If another Source is specified, the Event ID as listed earlier becomes invalid.

Lpr

CCM Core sends a spool file to any printer that understands the Line Printer Daemon protocol (Lpr). CCM Core cannot verify if the printer understands the data in the spool file.

The host must run an LPD daemon. This daemon must support the basic requirements as described in RFC 1179.

CCM Core does not support the optional switching of Control and Data files as described in section 6 of RFC 1179.

The Lpr protocol requires every client to submit its jobs using a unique 3-digit Job ID. Due to limitations in the Lpr protocol and the commonly used Berkeley implementation of the LPD daemon, CCM Core is not able to determine whether a Job ID conflicts with jobs pending on the host. Therefore, CCM Core uses a sequence file to generate sequential Job IDs. If there are multiple CCM Core services running

on the same NT server who can submit jobs using the `Lpr` command, we strongly advise you to use the SequenceFile(<text>) parameter to specify the same sequence file for every server. This file should be located in a shared directory that can be accessed by every service.

Syntax

```
Lpr
   File(<text>)
   Host(<text>)
   Printer(<text>)
   User(<text>)
   Banner(<text>)
   SequenceFile(<text>)
   Connection(<text>);
```

Parameters

- `File`: Required. The spool file that CCM Core sends to the printer.
- `Host`: Required. The host where CCM Core sends the spool file.
- `Printer`: Optional. The name of the printer that CCM Core prints to. This printer must be defined on the host. If this parameter is not specified, CCM Core sends the document to the default ("lp") printer.
- `User`: Optional. The User ID that CCM Core uses when submitting the job. If this parameter is not specified, CCM Core sends the document with the User ID "itpds."
- `Banner`: Optional. The text printed on the banner page. If this parameter is not set, CCM Core requests the host to suppress the banner page.
- `SequenceFile`: Optional. Name of a file that CCM Core uses to generate sequence numbers. Use this setting when more than one CCM Document Processor is going to print to the host. You can set the same file for all Document Processors as they share it.
- If this parameter is not set, CCM Core uses an automatically created sequence file in its installation directory, with one for each CCM Document Processor.
- `Connection`: Optional. The range of network ports that CCM Core uses to attempt to connect to the LPD server.

  The range should be specified as:

  - Connection ("x") to use only port x
  - Connection ("x:y") to use ports x through y (inclusive)

  The default value for this parameter is Connection ("721:731").

Note RFC 1179 restricts the allowed source port range to ports 721 to 731. CCM Core by default only uses this range as the LPD server is allowed to refuse connections outside this range. It is possible for servers to exhaust the available ports if many jobs are sent in a short time frame to the same LPD server. In this situation you can use the Connection(..) parameter to expand the range of available network ports.

You cannot specify a range accepted by any LPD server, but you can use the same port range used by the Microsoft Windows NT LPR client (513 - 1023) as it should be accepted by most LPD clients.

This alternative range can be specified as Connection ("513:1023").

Mail

CCM Core sends an email with optional attached documents to any valid recipient.

The SMTP host must support the minimum requirements as described in RFCs 821 and 822.

CCM Core uses MIME enhancements if the email contains 8-bit characters. Attachments are sent using MIME enhancements. MIME types for attachments are read from the Registry. If the MIME type cannot be read from the Registry, the file is sent binary with application/octet-stream as MIME type.

In general, you can use any Internet email gateway as a SMTP host.

To learn how to construct a valid email message, see Send email from a script.

Syntax

```
Mail
   File(<text>)
   Host(<text>)
   From(<text>)
   To(<text>)
   Attachments(<text>)
   Preformatted(<True or False>)
   Port(<text>)
   MimeEncoding(<text>);
```

Parameters

- `File:` Required. The header and MIME body of the email.
- The header in this file is the human readable header of the mail. Include `Subject:`, `To:`, and `Reply-to:` in this header. For more information, see Send email from a script. The body of this file is also sent, but is only shown if the recipient has an email client that does not support MIME enhancements or when no Attachments parameter is passed. It is customary that this file should contain a message informing the recipient that he or she needs MIME support to access the attachments.
- `Host:` Required. The SMTP host that CCM Core uses to send the email.
- `From:` Required. The User Name used to transmit the email. This user name must be specified in the user@hostname.domain format. CCM Core needs this user name to authenticate the sender to the SMTP host.
- `To:` Required. A string containing a comma-separated list of recipients of the email. Every recipient must be specified in the user@hostname.domain format.
- `Attachments:` Optional. A string containing a comma-separated list of documents that CCM Core attaches to the email message as MIME attachments. The first two of these attachments are the alternative bodies of the email using the rules described in the following table.

	File extension first file	File extension second file
Body alternative sent: TXT/HTML*	TXT	HTML
Body alternative sent: TXT/HTML*	HTML	TXT
Body alternative sent: TXT	TXT	Not HTML

*Which of the provided alternative bodies is shown depends on the setting of the mail client.

For more information on the MIME enhancement, see Send email with an attachment.

Inline image in HMTL Body

Inline images are supported (Content_ID header support). For more information, see HTML inline images.

`Preformatted`: Optional. If this parameter is set to Y, the document provided with the File (<text>) parameter is considered to contain a complete, formatted email, including all headers and attachments. The value of the parameter Attachments (<text>) is ignored in this case.

`Port`: Optional. The number or symbolic name of the port that CCM Core connects to on the host. If this parameter is not specified, CCM Core uses the standard SMTP port (port 25).

`MimeEncoding`: Optional. The encoding used to send the HTML part of the body. This setting must match the encoding used to generate the HTML page as it does not convert the page. Some mail clients use this attribute to determine the encoding instead of the appropriate metadata tags embedded within the HTML page.

Note The command `SimpleMail` offers a simpler interface for producing elementary email. For more information, see SimpleMail.

Error conditions

The `Mail` command fails if the mail server reports an invalid recipient or an empty recipient. In that case, the mail message is not sent. This is the safest approach as it does not ignore errors.

To send mail to a list of recipients, use a separate `Mail` command for each recipient and handle possible errors.

This command only validates recipients against the local mail server. It catches errors such as badly formatted email addresses, but it does not non-existent remote addresses. In other words, the validation does not guarantee that a message is delivered at the destination.

To send emails to a list of recipients where some entries in that list could be empty, use the following scripting to filter out the empty recipients out.

Filter empty recipient entries.

```
Var Text Rcpts = "";
Var Text Rcpt;
ForEach Rcpt In <original list> Do
  If Rcpts = "" Then
    Rcpts = Rcpt;
  Else
    Rcpts = Rcpts + "," + Rcpt;
  Fi;
Od;
```

The `ForEach` commands skip empty items and filter the list in the process. The original list must be comma-separated.

MergePDF

`MergePDF` merges the pages of two PDF documents with each other. This command can be used to merge `PDF` documents to create overlays and watermarks.

Syntax

```
MergePDF
   Src(<text>)
   Watermark(<text>)
   Dest(<text>)
   Repeat(<True or False>)
   Overlay(<True or False>)
   ProducePDFA(<True or False>);
   Processor("Amyuni" or "PDFLib");
```

Parameters

- `Src`: Required. The full path of the base PDF document.
- `Watermark`: Required. The full path of the watermark PDF document. The pages from this document are merged into the Src document either as overlays or as watermarks.
- `Dest`: Required. The full path of the resulting PDF document that contains the merged pages. If the source PDF file contains attachments, they are removed from the result PDF file.
- `Repeat`: Optional (Default setting: False). This parameter determines how the `MergePDF` command performs the merge if the Watermark PDF document has fewer pages than the Src PDF document. If the parameter Repeat is set to False, all pages of the Watermark PDF document are merged into corresponding pages of the Src PDF document and the remaining pages of the Src PDF document are copied unmodified. For example, applying the watermark on the first page of a document.
- If the parameter Repeat is set to True, all pages of the Watermark PDF document are merged into the corresponding pages of the Src PDF document. When the end of the Watermark PDF document is reached, the next page is again merged with the first page of the Watermark PDF document. Sample use 2: Watermark all pages of a document with the same watermark.
- `Overlay`: Optional (Default setting: False). This parameter determines how the Src and Watermark PDF documents are merged. If the parameter Overlay is set to False, the pages of the Watermark PDF document are inserted as a watermark, below the text of the Src document. If the parameter Overlay is set to True, the pages of the Watermark PDF document are inserted as an overlay, covering the text of the Src document.
- `ProducePDFA`: Optional (Default Setting: False). This parameter determines whether the resulting document is PDF/A-1b compliant.
- `Processor`: Optional. Selects the conversion technology used to concatenate the PDF files. If this parameter is set to "Amyuni", the Amyuni toolkit is used. If this parameter is set to "PDFLib", the PDFLib toolkit is used. Default is "Amyuni". You can change the default through the following setting in the dp.ini file:

  ```
  PDFTools.Processor=<"Amyuni" or "PDFLib">
  ```

Note The `MergePDF` command can only produce PDF/A-1b compliant documents if both the Src and Watermark documents are PDF/A-1b compliant and if the parameter `ProducePDFA` is set to True.

When selecting the conversion technology, note the following:

- Amyuni
  - `MergePDF` is only supported if both input documents are produced using `DocToPDF` with the Processor value set to "Word".
- PDFLib
  - `MergePDF` cannot transfer interactive content and document metadata from the documents.
  - As of CCM Core version 5.1.1, hyperlinks and bookmarks are transferred from the input document to the result.

Global settings

The following setting, when added to the [Configuration] section of the CCM Core installation dp.ini file, changes the behavior of the `MergePDF` command. This setting is specific for the PDFLib conversion technology and is ignored when not applicable.

`PDFLib.StrictParameterCheck`

Set to Y to have the `MergePDF` command report an error when it encounters an unsupported parameter or setting if `Processor` is set to `PDFLib`.

```
PDFLib.StrictParameterCheck=<Y or N>
```

If not set, it defaults to N.

Move

CCM Core moves the specified file.

The `Move` command fails if `Dest` already exists.

Syntax

```
Move
   Src(<text>)
   Dest(<text>)
   TimeOut(<number>);
```

Parameters

- `Src`: Required. The file that CCM Core moves.
- `Dest`: Required. The new name and directory where CCM Core moves the file to.
- `TimeOut`: Optional. The maximum amount of time in seconds that CCM Core waits if another process is locking `Src`. If this parameter is not specified, CCM Core waits indefinitely for the file to be unlocked. If the value of this parameter is 0 and `Src` is locked, CCM Core immediately reports an error.

OnError

`OnError` installs a script executed when a command fails. The script is loaded at the moment CCM Core is started. If an error occurs, the current script terminates and CCM Core transfers control to

the script `OnError`. CCM Core stores error information for up to ten error occurrences in a folder CompositionErrors in the Document Processor subfolder of the Temp folder. After ten occurrences, the previous information is overwritten again starting with the oldest error information.

If an error occurs and no error handler has been installed, the script terminates and a run-time error is reported.

There are two special script names:

- Script(*) forces CCM Core to ignore any errors and to continue.
- Script(-) forces CCM Core to cancel any previous `OnError` commands in the script and to terminate normally if an error occurs, as if no error handler has been installed.

Syntax

```
OnError
Script(…)
<Key>(…)
<Key>(…)… ;
```

Parameters

- `Script`: Required. The name of the script called whenever an error occurs.
- `<Key>`: Optional. Zero, one or more parameters passed to the script. The type of these keys is determined by the type of the parameters needed in the script called with the parameter Script(…).

PrintDocument

CCM Core instructs Microsoft Word to open the Src file to print this file to the specified printer. CCM Core automatically chooses the right processor by inspecting the contents of the Src file. The extension of the Src file is irrelevant. The format of the Src file should be one of the file formats that CCM Core recognizes.

Each processor has its own format for indicating a certain printer. Consequently, CCM Core has a special Printer key for every supported processor. Each of these keys should be used to indicate the printer using the format of the corresponding processor.

After CCM Core has chosen a particular processor, it passes the value of the appropriate printer key to it. If all printer keys are properly set, you can use a single command `PrintDocument` to print files from all supported processors to a particular printer. If you only need to print Src files of a single processor, you only need to specify the printer key of that particular processor.

Before you can use the word processors in CCM Core, they must be configured. For more information, see the chapter "Use of Microsoft Word by CCM Core" in the *Kofax Customer Communications Manager Installation Guide*.

Syntax

```
PrintDocument
   Src(<text>)
   File(<text>)
   Copies(<number>)
   Collate (<True or False>)
   TimeOut(<number>)
   WFWPrinter(<text>)
   PDFPrinter(<text>);
```

Parameters

- `Src`: Required. The file that CCM Core prints. Currently, the Microsoft Word and PDF formats are supported.
- `File`: Optional. If this parameter is specified, CCM Core prints to a file. The value of the parameter is the name of the resulting spool file. Any existing file is overwritten.
- `Copies`: If this parameter is specified, CCM Core requests the word processor to print the specified number of copies in a single print job. The number of copies should not exceed 32767. This parameter is currently only supported for Microsoft Word.
- `Collate`: Optional, Word only. (Default setting: False.) This setting only applies when a WFW printer is used (Microsoft Word). This setting allows you to print all pages of a copy before printing the first page of the next copy. Note that not all printer drivers support the Collate printing option of Word; the same can be said about duplex printers.
- `TimeOut`: Optional. The timeout for this command in seconds. If the word processor exceeds this timeout, the process is terminated and CCM Core reports a run-time error. If this parameter is omitted, the appropriate default timeout value for either batch or interactive jobs is used. A timeout value of 0 disables the timeout. This feature is currently supported for documents printed using Microsoft Word and PDF documents.
- `WFWPrinter`: Optional (Needed if Src is a Microsoft Word document). The name and port of the printer if CCM Core uses Microsoft Word to print Src. If this parameter is not specified, CCM Core uses the value of the constant WFWPrinter. If CCM Core uses Word and this constant is not set, CCM Core reports a run-time error.

  The printer name should be either the name of the printer and its port for local printers, or the name of the printer queue for remote printers.

  Example

  Use the following command to print to the PS printer that is defined locally.

  ```
  WFWPrinter("PS on LPT1:")
  ```

  Use this command to print to the `REMOTE` printer queue on `SERVER`.

  ```
  WFWPrinter("\\SERVER\REMOTE")
  ```

- `PDFPrinter`: Optional (needed if SRC is a PDF document). The name of the printer. If this parameter is not specified, CCM Core uses the value of the constant PDFPrinter.

  The printer name should be either the name of the printer for local printers or the name of the printer queue for remote printers.

Limitations and known issues when printing PDF documents:

- Only supported for PDF documents created with the `DocToPDF` command for which the Processor parameter value was "Word".
- Printing large documents (more than a few hundred pages) may lead to resource problems.
- You cannot print to a file.
- The collate flag is not supported.
- Some (embedded) PDF fonts do not print correctly in combination with certain printer driver settings. These problems may arise with PDF documents generated with older versions of CCM Core. If the print shows incorrect glyphs, the following may solve the issue: open the printing preferences of the driver for the account that CCM Core runs in. Click **Advanced** and clear the following settings under **Document Options** (if present): **Advanced Printing Features** and **Print Optimizations** (typically found on PCL drivers).

Example

Use this command to print to the PS printer that is defined locally.

```
PDFPrinter("PS")
```

Use this command to print to the PS printer that is defined remotely.

```
PDFPrinter("\\SERVER\REMOTE")
```

Microsoft Word 2003 compatibility mode

Printing Microsoft Word documents in the DOCX format can cost a significant amount of time when these documents contain graphics. Performance can be improved by forcing Microsoft Word 2010 and higher version to use the Microsoft Word 2003 Compatibility Mode. This mode uses an older version of the graphics engine built in Microsoft Word, which is also used for DOC documents.

CCM Core can be configured to force this compatibility mode by adding the following setting to the [Configuration] section of the dp.ini file:

```
Word2003CompatibleMode=Y
```

Forcing the compatibility mode can have an effect on the quality of the graphics and other content of the document. This setting has no effect on Microsoft Word 2007 or older versions. This setting was introduced in CCM Core version 4.2.3.

Progress

CCM Core sends a Progress Message to a synchronous client. This command has no effect when an asynchronous client has submitted the request.

Syntax

```
Progress
Message(<text>);
```

Parameters

`Message`: Required. The text of the message sent to the client.

Note The `Progress` command is also used to in combination with the Web Services interface to allow for communication between the caller and the CCM Core script. For details, see the section "SubmitEx" in the *Kofax Customer Communications Manager Core Developer's Guide*.

PStoPDF

This command uses Acrobat Distiller to create a PDF file from a PostScript file. Adobe Acrobat Distiller is a third party product, which needs to be installed on the machine containing the Document Processor that will process the job. Adobe Acrobat Distiller is **not** included with CCM Core.

CCM Core instructs Acrobat Distiller to open the Src document and to save this document to the file Dest as a PDF file. The Src document must be a PostScript file. The extension of the Src document does not matter. CCM Core logs the start/stop events of Acrobat Distiller if the LogLevel is set to level 2 or higher (progress). It also logs other messages from Acrobat Distiller if the LogLevel is set to level 3 or higher (info).

Note The Distiller command fails if an Adobe Distiller is active running under another user account than the account used for CCM Core and the Document Processors.

Syntax

```
PStoPDF
  Src(<text>)
  Dest(<text>)
  OptionsFile(<text>)
```

Parameters

- `Src`: Required. The PostScript file that CCM Core converts to PDF.
- `Dest`: Required. The name of the resulting PDF file. Any existing file is overwritten.
- `OptionsFile`: Optional. Specifies the name of an Acrobat Distiller job options file. Such a file can be created by configuring Acrobat Distiller and by saving the settings to a file. By passing a job options file, you can convert a file to PDF with specific job options.

  Note If this parameter is omitted, `PStoPDF` uses the current distiller settings. Also, the job options file only contains the Job Options settings located at **Acrobat Distiller** > **Settings**. Security settings cannot be specified through the Job Options files.

ReceiveFile

The `ReceiveFile` command is used to download binary data from the client to CCM Core. This feature requires a synchronous TCP/IP connection or an MQSeries interface.

For MQSeries, this command reads a message from the Src queue and stores its contents in the file Dest. It reads the next message with the correlation ID matching the message ID of the request. For details, see "MQSeries protocol" in the *Kofax Customer Communications Manager Core Developer's Guide*.

Syntax

```
ReceiveFile
   Src(<text>)
   Dest(<text>)
   TimeOut(<number>);
```

Parameters

- `Src`: Required. For TCP/IP, a client side file identification. For MQSeries, the name of a queue.
- `Dest`: Required. The file name on the CCM Core side.
- `TimeOut`: Optional. The maximum amount of time in seconds the command `ReceiveFile` waits for the download to complete. If this time is exceeded, the download is aborted and an error is reported.

If this parameter is omitted, CCM Core uses the default timeout interval as configured in CCM Core Administrator. If this parameter is set to 0, the download does not time out.

Remarks

For TCP/IP, the `ReceiveFile` command does not perform code page translations. For more information, see ConvertCodepage.

For MQSeries, it depends on the format of the message whether CCM Core Services try to convert data. If the format of the data message is set to MQFMT_STRING, CCM Core tries to convert the data to Unicode before storing it in the indicated file. Otherwise, it requests no conversion. In the latter case, whether no conversion takes place depends on the configuration of the MQSeries queues, the format of the message, and on whether the client demands a conversion when putting the data on the queue. For more information, see "QSeries interface" in the *Kofax Customer Communications Manager Core Developer's Guide*.

If the command `ReceiveFile` is terminated due to a timeout the connection to the CCM Core process is reset to ensure that data in transit is discarded and that processes resynchronize correctly. This forced reset can result in additional network-related errors in the log for this job. Any further communication from this job fails between the CCM Document Processor and the client. The client will be informed that the CCM Document Processor is disconnected.

RemoveDirectory

CCM Core removes the specified directory. When the Recursive parameter is True, the folder and all of its contents are removed; otherwise, only empty directories are removed.

- The `RemoveDirectory` command succeeds if the specified directory could be removed or if the directory did not exist.
- The `RemoveDirectory` command fails if the specified directory is not empty, and the parameter Recursive is not True.
- The `RemoveDirectory` command fails when the specified folder cannot be deleted, or a file or a directory within the specified folder cannot be deleted when the parameter Recursive is True.

Syntax

```
RemoveDirectory
   Dir(<text>)
   Recursive(<True or False>);
```

Parameters

- `Dir`: Required. The directory that CCM Core removes.
- `Recursive`: Optional (the default value is False). If set to True, CCM Core removes all directories and files in the specified directory, in addition to the directory itself. This parameter is used to recursively remove a directory tree.

RepositoryImportProject

Use this call to import a CCM Repository project.

Note Another method to import CCM Repository projects, RepImport.exe, is deprecated. We recommend that you use RepositoryImportProject instead.

Syntax

```
RepositoryImportProject
    ProjectName (<text>)
    FilePath (<text>)
    Overwrite (<Boolean>);
```

Parameters

- `ProjectName`: Required. The name of the project to import. Characters not supported for a project name are removed.
- `FilePath`: Required. The project file to import.
- `Overwrite`: Optional. If set to False, the import will fail if the project with the given name already exists. If set to True, the import will overwrite the existing project. In either case, the project is created if it does not exist yet. Default is False.

Consider the following when importing project exports:

- When importing a regular or "Archive and helpdesk" project export, the import fails if the project with the same name already exists and contains active Changesets created and controlled by an external workflow. To replace an existing project when importing the export, deactivate or publish such Changesets in the existing project.

  For more information on an "Archival and helpdesk" export, see the *Kofax Customer Communications Manager Repository User's Guide*. For more information on Changesets, see the CCM Designer online Help.
- If you are overwriting an existing project, the import replaces any Changesets from the existing project. A regular export cannot contain Changesets, so after you import it replacing an existing project, no Changesets are present in the final project. An "Archival or helpdesk" export may contain Changesets, and after you import it replacing an existing project, the final project only contains the Changesets that are present in the export.

RepositoryExportProject

Use this call to export a CCM Repository project.

Note Another method to export CCM Repository projects, RepExport.exe, is deprecated. We recommend that you use RepositoryExportProject instead.

Syntax

```
RepositoryExportProject
    ProjectName (<text>)
    FilePath (<text>);
```

Parameters

- `ProjectName`: Required. The name of the project to export. Characters not supported for a project name are removed.

- `FilePath`: Required. The project file to export.

RestoreSession

CCM Core creates a new session and loads its state from a previously saved session. For more information on sessions, see CCM Core sessions.

The command `RestoreSession` fails if the active job has already been associated with a session.

Syntax

```
RestoreSession
   Archive(<text>)
   RequireEncryption(True or False);
```

Parameters

- `Archive`: Required. Provides the name of a file in which the state of the session was saved earlier using the command `SaveSession`.
- `RequireEncryption`: Optional. Indicates whether a non-encrypted session archives can be restored. If this parameter is set to True, only encrypted archives are accepted.

Remarks

The command `RestoreSession` restores the session parameters and files in the directory _sessiondir. You can restore a saved session more than once. Each restored session receives a different unique session identifier.

If the command `RestoreSession` completes successfully, the variable _restored_session contains the original session identifier of the saved session.

Note The command `RestoreSession` validates the validity of an archive before restoring the session. If the archive fails this check, an error is issued and the command `RestoreSession` fails.

RetrieveRepositoryObject

CCM Core retrieves a dynamic object from the specified CCM Repository. The dynamic object is stored in a file.

You can only use `RetrieveRepositoryObject` in the context of a Session.

Syntax

```
RetrieveRepositoryObject
  URI (<rep:/ uri>)
  Result (<file>)
  Encoding (<format>);
```

Parameters

- `URI`: Required. A rep:/ URI that specifies the object that should be retrieved.

- `Result`: Required. The file in which the retrieved object should be stored. If this file already exists, it is overwritten.
- `Encoding`: Optional. If this parameter is omitted or set to "native," the object is stored in its native format. If this setting is set to "base64," the object will be base-64 encoded before it is stored. If this setting is set to "source-document," the word processor source variant of the object is stored. This feature is currently only supported for Rich Text Blocks; for other objects encoding "source-document" is treated as "native."

Rep:/ URIs

The URIs supported by `RetrieveRepositoryObject` use the following generic format.

```
rep:[//host[:port]]/type/project/[path/]object[?key=value[&key=value]*]
```

`host`: TCP/IP Host name of the system hosting the CCM Repository server.

`port`: TCP/IP Port the server listens to (defaults to 2586)

`type`: Type object to retrieve. Supported objects are:

- `textblock`
- `mastertemplate`
- `wizard`
- `form`
- `documentationdocument` (any object in the CCM Repository Documentation folder)
- `styledocument`
- `quickdocument`
- `letterbook`
- `staticdocument`
- `resource` (any object in the CCM Repository Resources folder)
- `documentpacktemplate`

`project`: Project. Use "*" to refer to the project. The previous project refers to values entered in previous calls to `RetrieveRepositoryObject` in the same session. Once objects are retrieved from a specific project you are not allowed to switch to other projects in the same session. This refers to changes in the values for host: port and project as entered in the URL. Changing these values returns an error.

`path`: Optional. Folders, separated by a slash.

`object`: The object to be retrieved.

`key/value`: Additional key/value pairs for parameters. Supported keys are:

- `user=` Repository User. This is only relevant when development status objects are retrieved.
- `status=[published|accepted|current|development]`. If the status is omitted, the published status is retrieved.

When retrieving objects from CCM Repository using `RetrieveRepositoryObject`, the following results can be expected.

Type	Result

`textblock`	Returns the Text Block XML for the Text Block. For Rich Text Blocks, setting the `Encoding` parameter to "source-document" returns the Microsoft Word document defining the Rich Text Block instead.
`mastertemplate`	Returns the compiled Master Template. This is not a Microsoft Word document.
`wizard`	Return the Content Wizard XML describing the Content Wizard
`form`	Returns the Form XML describing the Form
`documentationdocument`	Returns the documentation object requested. The type is equal to the type of the requested object.
`styledocument`	Returns the Microsoft Word document containing the styles
`quickdocument`	Returns the compiled Quick Template. This is not a Microsoft Word or HTML document.
`letterbook`	Returns the Letter Book XML describing the Letter Book
`staticdocument`	Returns the static document requested. The type is equal to the type of the Static Document and is either a .doc, .docx, .pdf, .xls, .xlsx, .ppt, or .pptx.
`resource`	Returns the resource object requested. The type is equal to the type of the resource.
`documentpacktemplate`	Returns the Document Pack Template XML describing the Document Pack Template

Examples

```
rep://localhost:2587/letterbook/InstallationTest/Letters?status=accepted
```

This example refers to the accepted version of the Letter Book `Letters` in the project `InstallationTest`. The `letterbook` is retrieved from the CCM Repository server installed on the `localhost` for the instance indicated by port `2587`. For information on the host and port to use for each instance, see the ReadMe.txt file that resides in: `<instance folder>\designer\Client`.

```
rep:/textblock/*/Clauses/US/Clause421
```

This example refers to the Text Block `Clause421` in the Text Block folder `Clauses/US`. The location of the Repository server (host:port) and the project as indicated by * has been omitted and defaults to the previously used resources in the session.

Metadata

Some objects can provide additional information. This information is available in the internal constant _repository_metadata. If there is no information available, this constant is empty.

If the metadata contains multiple parts, they are separated by the content of the internal constant _newline.

`textblock`: _repository_metadata contains "itptbkxml<newline>description" for normal Text Blocks and "rich<newline>description" for Rich Text Blocks. Any line breaks within the description are replaced by spaces.

`static document`: _repository_metadata contains the extension of the static document.

Example

```
RetrieveRepositoryObject
  URI ("rep:/letterbook/myProject/Letters")
  Encoding ("Base64")
  Result ("letterbook" [ _SessionDir, "xml" ]);
```

This example retrieves the Letter Book `Letters` from the project `myProject`. The Letter Book definition is Base-64 encoded and stored in the file letterbook.xml in the current session directory.

Availability

The `RetrieveRepositoryObject` command is introduced in CCM Core 4.4.

RunCommand

CCM Core starts an external program and waits for the program to terminate. The command `RunCommand` fails if the external program terminates with a non-zero return code. The output of the command can be captured into both the CCM Document Processor logs and into script variables for further processing.

Syntax

```
RunMacro
   Document(<text>)
   Macro(<text>)
   Arg1(<text>)
   Arg2(<text>)
   ...
   Arg10(<text>)
   TimeOut(<number>);
```

Parameters

- `CmdLine`: Required. The command line that is executed.
- `LogStdOut`: Optional. If this parameter is set to True, all output written by the command to its "standard output" channel is added to the CCM Document Processor log. The log level for the CCM Core should be at least three (Extended info). If this parameter, is omitted or set to False, no output is written to the logs.

  Note Most command line tools buffer their output into blocks before writing it to the "standard output" channel. It is therefore possible that lines are broken into multiple parts when written to the log.

- `LogStdErr`: Optional. If this parameter is set to True, all output written by the command to its "standard error" channel is added to the CCM Document Processor log. The log level for the CCM Core should be at least three (Extended info). If this parameter is omitted or set to False, no output is written to the logs. Most command line tools write their error messages to the "standard error" channel.
- `UnicodeOutput`: Optional. If this parameter is set to True, all output of the command is interpreted as Unicode text. If this parameter is omitted or set to False, the output is treated as single-byte text in the local code page of the system.
- `CollectOutput`: Optional. If this parameter is omitted or set to True, all output written to the "standard output" channel and "standard error" channel are captured and made available to the script

in the variables _stdout and _stderr, after the `RunCommand` completes. If this parameter is set to False, the channels are not captured and the variables _stdout and _stderr are empty.

You can use `RunCommand`/`StartProgram` to execute scripts, such as batch and .vbs files, by using the appropriate script processor to execute these scripts. For example, to run a CMD file:

```
RunCommand
CmdLine("cmd /c c:\scripts\do_work.cmd par >
c:\logs\out.txt");
```

This command redirects the "standard output" channel to the file *c:\logs\out.txt.* Therefore, the variable _stdout does not capture anything and is empty when the command is completed.

The _stdout and _stderr variables are by default only capture the first 100 lines of output. If there is more output, the variable ends with the line "TRUNCATED." The number of lines can be changed by adding the following setting to the [Configuration] section of the dp.ini of your CCM Core setup. Replace 100 with the required number of lines, or 0 to remove any limitation.

```
CaptureLineLimit=100
```

Important If the program generates an excessive amount of output (in the order of hundreds of megabytes), the CCM Document Processor can shut down if it runs out of memory to store the text. In such a scenario, write the output to disk and preprocess it with external tools before reading it into the CCM Core script.

Logging to the CCM Document Processor log file, whenever LogStdOut or LogStdErr is enabled, is never truncated. This logging is subject to the normal log rotation schedule.

Some applications may become unresponsive when their output is captured. Disable capturing when calling the following program:

- PsExec.exe (SysInternals PsTools)

RunMacro

This command is only supported for Microsoft Word. It instructs Microsoft Word to open "Document" and run the specified macro. Microsoft Word executes the macro from the currently loaded templates and documents.

The macro must save and close any opened document, including the "Document," before terminating.

Interaction with the user is not possible, and the macro must provide non-interactive error handling.

Syntax

```
RunMacro
   Document(<text>)
   Macro(<text>)
   Arg1(<text>)
   Arg2(<text>)
   ...
   Arg10(<text>)
   TimeOut(<number>);
```

Parameters

- `Document`: Required. Microsoft Word document that CCM Core loads. Currently only Microsoft Word is supported. The macro should save and close this document.
- `Macro`: Required. The name of the macro that will be run.
- `Arg1`: Optional. First parameter for the macro.
- `Arg2...Arg10`: Optional. Second through tenth parameter for the macro. Allow the script to pass up to ten parameters to the macro.
- `TimeOut`: Optional. The timeout for this command in seconds. If the WordProcessor exceeds this timeout, the process is terminated and CCM Core reports a run-time error. If this parameter is omitted, the appropriate default timeout value for either batch or interactive jobs is used. A timeout value of 0 disables the timeout.

Word macros

Example

```
RunMacro
   Document("C:\temp\fox.doc"))
   Macro("MyProject.MyModule.MyMacro")
   Arg1 ("First parameter")
   Arg2 ("Second parameter")
   Arg3 ("Third parameter")
   Arg4 ("Fourth parameter");
```

SaveDocumentPack

Use the command `SaveDocumentPack` to write the active Document Pack to a file. This command generates or updates the manifest.xml file for the Document Pack.

The command `SaveDocumentPack` fails if there is no active Document Pack in the session.

Syntax

```
SaveDocumentPack
 File (<text>);
```

Parameters

`File`: Required. The file to save the Document Pack into.

SaveSession

CCM Core saves the state of the session associated with the job and all files stored in the directory _sessiondir and its subdirectories into an archive file. The session is removed after its state has been saved. For more information on sessions, see CCM Core sessions.

The command `SaveSession` fails if the active job has not been associated with a session or if the archive file already exists.

Syntax

```
SaveSession
   Archive(<text>)
   Encrypt(True or False);
```

Parameters

- `Archive`: Required. Provides the name of the file in which the state of the session is saved.
- `Encrypt`: Optional. Indicates if the contents of the session archive should be encrypted. If this parameter has the value True, the contents of the session archive are stored in an encrypted format. If this parameter is omitted or has the value False, the session data is saved in a ZIP file and can be viewed with conventional tools.

Remarks

The command `SaveSession` saves the session parameters and all files from the directory _sessiondir. After the command is completed, the current session is cleaned up, and the current job is no longer associated with a session.

If applications running in the background have a lock on one of the files in the directory _sessiondir, the archive may fail.

Note The `SaveSession` command allows the administrator to compress and encrypt the session archive to prevent access to sensitive content if the archive is uploaded to the user (as used in the CCM ComposerUI Server sample implementation). The archive has also a checksum which is used to validate the contents before restoring the session. It is allowed to add data to the end of the archive without invalidating its contents. The command `RestoreSession` ignores this additional data.

SecurePDF

You can use the command `SecurePDF` to digitally sign, encrypt, and set access controls on PDF documents. The digital signature is based on a cryptographic certificate, and you can use it to validate the authenticity of the PDF document. Any changes made to the PDF document after it has been signed invalidate the signature. Encryption restricts access to the protected document to users who have the appropriate password. The access controls restrict the ways in which the content of the PDF document can be manipulated by the reader.

The `SecurePDF` command does not produce PDF/A-1b compliant documents, even when the input PDF document is PDF/A-1b compliant.

Syntax

```
SecurePDF
        File(<text>)
        CertificateName(<text>)
        AllowPrint(<True or False>)
        AllowCopy(<True or False>)
        AllowChange(<True or False>)
        AllowFieldAuthoring(<True or False>)
        PermissionPassword(<text>)
        OpenPassword(<text>)
        Use40BitEncryption(<True or False>);
        Processor("Amyuni" or "PDFLib");
```

Parameters

- `File`: Required. The path and file name of the PDF file. This must be a PDF file on which none of the restrictions is set. If the PDF file contains attachments, they are removed.
- `CertificateName`: Optional. (Default: the PDF file is secured, but not signed.) The name of the stored certificate used for signing the PDF file. The certificate is issued to this name.
- The certificate must be installed in the Personal certificate store associated with the CCM Core account.
- `AllowPrint`: Optional (Default: False). Specifies whether the file can be printed by users.
- `AllowCopy`: Optional (Default: False). Specifies whether the document content can be copied by users.
- `AllowChange`: Optional (Default: False). Specifies whether the document content can be changed by users.
- `AllowFieldAuthoring`: Optional (Default: False. Users can still fill out the fields). Specifies whether Form Fields and comments can be added or changed.
- `PermissionPassword`: Optional (Default: a random password, unknown to all). Specifies the password required to change access restrictions for the PDF file set with this command. If you do not specify a permission password, a random password unknown to all is used.

  Note If no permission password is specified, you cannot change the permissions later through the Adobe Acrobat software. The permission password alone should not be regarded as a strong security measure. Most common software, such as Adobe Acrobat, respects the PDF security settings. But you can develop and obtain software tools that remove or ignore the access restrictions for PDF documents that do not have an open password. You can find information on the parameter `OpenPassword` and security under "Security remarks" later in this section.

- `OpenPassword`: Optional (Default: no password is needed to open the PDF). If a non-empty string is given, this parameter is the password that must be used to open the document. If a password is set, all access to the document is prohibited without it.

  The default is empty, which means that anyone can open the document and at least view it.

  Note If you specify an `OpenPassword` and an empty `PermissionPassword`, there will be no access restrictions once a user opens the document with the open password.

  An open password should not be regarded as a strong security measure in combination with an empty `PermissionPassword`. The `SecurePDF` parameter reduces this risk by generating a random permission password by default. The Adobe software does not allow to open a file without specifying `OpenPassword`, but you can develop/obtain software tools that remove/ignore the open password for PDF files that have an empty (non-random) permission password.
- `Processor`: Optional. Selects the conversion technology used to concatenate the PDF files. If this parameter is set to "Amyuni", the Amyuni toolkit is used. If this parameter is set to "PDFLib", the PDFLib toolkit is used. Default is "Amyuni". You can change the default through the following setting in the dp.ini file:

  ```
  PDFTools.Processor=<"Amyuni" or "PDFLib">
  ```

Also, when selecting the conversion technology, note the following:

- **Amyuni**
  - `SecurePDF` uses the RC4 cipher with a 128-bit key to encrypt the PDF file if the OpenPassword is specified.
- **PDFLib**
  - `SecurePDF` uses the AES cipher with a 128-bit key to encrypt the PDF file if the OpenPassword is specified.
  - Signing the PDF file with a certificate is not supported.
  - `SecurePDF` removes interactive and document metadata from the documents.
  - As of CCM Core version 5.1.1, hyperlinks and outlines are retained in the document.

Security remarks

- The parameter `OpenPassword` is used to encrypt the PDF document using the 128-bit RC4 stream cipher. For optimal safety, use passwords of around 32 characters.
- The most secure file is achieved when the parameter `OpenPassword` is specified and the parameter `PermissionPassword` is not specified. The effect of the combination of these parameters is that users need a password to open the file and that the parameter `PermissionPassword` is unknown, so the security cannot be changed.
- The second most secure file is achieved with both the parameters `PermissionPassword` and `OpenPassword` set to different non-empty values. The effect of this combination is that users need a password to open the file, and that they need to know the `PermissionPassword` to change the security settings. If you use the same password for both opening and permissions, users are able to change the security settings after providing the open password alone.
- Either specifying an empty `PermissionPassword` or specifying an empty `OpenPassword` (or omitting it) leads to a less secure file. The file will be encrypted and have access restrictions respected by the Adobe Acrobat software.
- The security level that can be reached with the parameter `SecurePDF` depends on the passwords set and on the 128-bit RC4 stream cipher standard and as such needs to be assessed for adequacy before use.

`Use40BitEncryption`: This setting is ignored. Documents are always encrypted using 128- bit encryption.

Global settings

The following setting, when added to the [Configuration] section of the CCM Core installation dp.ini file, changes the behavior of the `SecurePDF` command. This setting is specific for the PDFLib conversion technology and is ignored when not applicable.

`PDFLib.StrictParameterCheck`

Set to Y to have the `SecurePDF` command report an error when it encounters an unsupported parameter or setting if `Processor` is set to `PDFLib`.

```
PDFLib.StrictParameterCheck=<Y or N>
```

If not set, the `CertificateName` option is silently ignored and the resulting document is not signed.

SendFile

You can use the command `SendFile` to upload binary data from CCM Core to the client. This feature requires a synchronous TCP/IP connection or an MQSeries interface.

For MQSeries, this command reads the contents of the file Src and sends it in a message to the queue Dest. It sets the correlation ID of the message to the message ID of the request. For a detailed explanation, see "MQSeries protocol" in the *Kofax Customer Communications Manager Core Developer's Guide*. If the script specifies *REPLYQ as the Dest queue, the MQSeries client puts the message on the queue specified in the attributes ReplyQ/ReplyQMgr of the request message.

Syntax

```
SendFile
   Src(<text>)
   Dest(<text>)
   TimeOut(<number>);
```

Parameters

- `Src`: Required. File on the CCM Core side.
- `Dest`: Required. For TCP/IP, a client side file identification. For MQSeries, the name of a queue.
- `Timeout`: Optional. The maximum amount of time in seconds the command `SendFile` waits for the upload to complete. If this time is exceeded, the upload is aborted and an error is reported. If this parameter is omitted, CCM Core uses the default timeout interval as configured in CCM Core Administrator. If this parameter is set to 0, the upload does not time out.

Remarks

The command `SendFile` does not perform code page translations. For more information, see ConvertCodepage.

If the command `SendFile` is terminated due to a timeout, the connection to the CCM Core process is reset to ensure that data in transit is discarded and processes resynchronize correctly. This forced reset can result in additional network-related errors in the log for this job. Any further communication from this job between the CCM Document Processor and the client fails. The client is informed that the CCM Document Processor is disconnected.

SetSessionParameter

You can use the command `SetSessionParameter` to assign a value to a session parameter. Use session parameters to store values between jobs associated with the same session. The stored values are retrieved using the function `get_sessionparameter`. For more information, see CCM Core sessions.

Note `SetSessionParameter` is implemented as a script component and is part of the built-in CCM Core script library. The script for `SetSessionParameter` is available in: `<deploy root>\CCM \Documentation\5.2\Resources\Examples\Core Scripting`. For more information, see Examples of script components.

Syntax

```
SetSessionParameter
   Par(<text>)
   Value(<text>);
```

Parameters

- `Par`: Required. The name of the session parameter.
- `Value`: The value that should be stored in the session parameter.

Example

```
SetSessionParameter
  Par ("plugh")
  Value ("plover");
Var Text X = get_sessionparameter ("plugh")  # returns "plover"
```

Remarks

The command `SetSessionParameter` is an interface to the function `set_sessionparameter` that does not return a value. The command `SetSessionParameter` and the function `set_sessionparameter` provide equivalent functionality.

Session parameters with names starting with _itp and ITP are reserved for internal use in CCM products.

SetTimeOut

CCM Core adjusts the timeout interval for the current job. The remainder of the job is subject to the new timeout parameters and is terminated if the timeout interval is exceeded.

Syntax

```
SetTimeOut
   TimeOut(<number>)
   GracePeriod(<number>);
```

Parameters

- `TimeOut`: Required. The timeout interval in seconds allowed for the job to complete. If this parameter is set to 0, the timeout interval is disabled and the job is allowed to run indefinitely.
- `GracePeriod`: Optional. The amount of time in seconds the job is allowed for termination of the active statement and possible error recovery. When this additional time is exceeded, the CCM Document Processor is forcibly restarted. If this parameter is set to 0, the CCM Document Processor is restarted immediately when the timeout interval expires.

Remarks

The default timeout intervals for jobs are configured in CCM Core Administrator. This statement allows jobs more detailed control over their behavior.

Shutdown

This command shuts down CCM Core and it must be the last command in a script.

Syntax

```
Shutdown
   Message(<text>);
```

Parameters

`Message`: Optional. The error message sent back to the client as an error. If this parameter is omitted, CCM Core is shut down silently and reports success to the client.

SimpleMail

The command `SimpleMail` offers a simpler interface compared to the command `Mail`. This command offers parameters commonly used by email clients.

`SimpleMail` is implemented as a script component and is part of the built-in CCM Core script library. The script for `SimpleMail` is available in: `<deploy root>\CCM\Documentation\5.2\Resources\Examples\Core Scripting`. For more information, see Examples of script components.

Usage

```
SimpleMail
To (<text>)
From(<text>)
File (<text>)
Host (<text>)
Port (<number>)
CC (<text>)
BCC (<text>)
ReplyTo (<text>)
Subject (<text>)
MimeEncoding (<text>)
Attachments (<text>);
```

Parameters

- `To`: Required. Comma-separated list of recipients.
- `From`: Required. The email address of the sender.
- `File`: Required. File containing the body of the email message. This file must contain ASCII text and have the .txt extension.
- `Host`: Required. The SMTP host.
- `Port`: Optional. Port on which the SMTP host is contacted. Defaults to the standard SMTP port (port 25).

- `CC`: Optional. Comma-separated list of recipients who receive a carbon copy of the email. These recipients are listed in the header on the CC: line. Defaults to an empty list.
- `BCC`: Optional. Comma-separated list of recipients who receive a blind carbon copy. These recipients are not listed in the header. Defaults to an empty list.
- `ReplyTo`: Optional. The email address to which replies should be directed. Defaults to From.
- `Subject`: Optional. Subject line of the email. Defaults to an empty line.
- `MimeEncoding`: Optional. The encoding used to send the HTML part of the body. This setting must match the encoding used to generate the HTML page, but it does not convert the page. Some mail clients use this attribute to determine the encoding instead of the appropriate metadata tags embedded within the HMTL page.
- `Attachments`: Optional. Defaults to an empty list. A string containing a comma-separated list of documents that CCM Core attaches to the email message as MIME attachments. The first two of these attachments are the alternative bodies of the email using the rules described in the following table.

	File extension first file	**File extension second file**
Body alternative sent: TXT/HTML*	TXT	HTML
Body alternative sent: TXT/HTML*	HTML	TXT
Body alternative sent: TXT	TXT	Not HTML

* The mail client settings determine which alternative is displayed.

For more information, see Send email with an attachment.

Inline images are supported (Content_ID header support). See HTML inline images.

Error conditions

The command `SimpleMail` fails if the mail server reports an invalid recipient. In that case, the mail message is not sent. This is the safest approach as it does not ignore errors.

To send mail to a list of recipients, use a separate `SimpleMail` command for each recipient to handle possible errors.

The command `SimpleMail` only validates recipients against the local mail server. This catches errors such as badly formatted email addresses, but it does not catch non-existent remote addresses. In other words, the validation does not guarantee that a message is delivered to the destination.

SpoolFile

CCM Core sends a spool file to any printer accessible through the Windows NT printer system. This spool file is sent to the printer as is. CCM Core cannot verify if the printer understands the data in the spool file.

To generate a spool file, use the `PrintDocument` command and print to file (file parameter specified).

Syntax

```
SpoolFile
   File(<text>)
```

```
   Printer(<text>);
```

Parameters

- `File:` Required. The spool file that CCM Core sends to the printer.
- `Printer:` Required. The name of the printer that CCM Core sends the spool file to. This name can be any printer that Windows NT can access.

StartProgram

CCM Core starts the specified external program. CCM Core does not wait for the external program to finish.

Syntax

```
StartProgram
   CmdLine(<text>);
```

Parameters

`CmdLine:` Required. The command line executed.

StopAllDMs

CCM Core requests all currently connected CCM Document Processors to stop their data manager process.

Syntax

```
 StopAllDMs;
```

Remarks

Every CCM Document Processor runs a data manager process that can cache connections to database resources. Use this command to ensure that all cached connections to a database are closed by shutting down these processes. The data managers are automatically restarted on the next job that requires database access.

You can use the exit points DailyTask and HourlyTask to schedule this command from within CCM Core.

The command `StopAllDMs` uses the program saclient to submit a job to all CCM Document Processors. If the CCM Document Processors are hosted on different computers, the IP addresses of all computers must be allowed to submit requests to CCM Core. In the default configuration, CCM Core allows requests from remote computers.

SubmitMaitenanceJob

Submits a maintenance job to CCM Core. This job is run on every CCM Document Processor connected to CCM Core when the job is submitted.

Syntax

```
SubmitMaintenanceJob
   Service(<text>)
   Parameters(<text>)
   JobID(<text>);
```

Parameters

- `Service`: Required. Name of the service being requested.
- `Parameters`: Optional. List of parameters for the service. This list is processed as a list of command line parameters. If a parameter contains spaces, it must be enclosed within escape quotes.
- `JobID`: Optional. Job ID used to submit the job. If this parameter is omitted, the Job ID of the job calling the command `SubmitMaintenanceJob` is used.

Remarks

The script `SubmitMaintenanceJob` uses the program saclient to submit a job to all CCM Document Processors. If the CCM Document Processors are hosted on different computers, the IP addresses of all these computers must be allowed to submit requests to CCM Core. In the default configuration, CCM Core allows requests from remote computers.

Temporary

Registers a file as a temporary file for the script.

CCM Core removes a file when it executes the `Temporary` command to prevent conflicts if the script creates the file later. At the end of the current script, CCM Core tries to remove the file again to provide proper cleanup.

Syntax

```
Temporary
   File(<text>)
   EndOfJob(True or False);
```

Parameters

- `File`: Required. The name of the file that CCM Core manages as a temporary file.
- `EndOfJob`: Optional. If this parameter is not present or if its value is False, CCM Core deletes the file at the end of the current script. If this parameter is True, CCM Core deletes the file at the end of the job.

ThrowError

CCM Core forces an error.

The command `ThrowError` puts CCM Core in the error state. The current request terminates unless an earlier `OnError` command has defined an error handler.

You can use the command `ThrowError` at the end of an error handler to report the final error after cleanup.

Syntax

```
ThrowError
   Message(<text>);
```

The `Message` is logged by CCM Core as an informational message. The command `ThrowError` is not logged as an error.

Parameters

`Message`: Required. The message is sent back to a synchronous client. If this parameter is not specified, CCM Core sends back the last real error message.

[@*USER] prefix

This section is relevant for users working on scripts that aim at supporting interfaces defined by the Contract Manager.

In some cases, if the request from the Contract Manager to CCM Core results in an error, the error is logged by the Contract Manager, and an informational message that refers to the log is returned to the client. To instruct the Contract Manager to pass the error message as is, add the prefix `[@*USER]` to the message text, as shown in this example.

```
ThrowError
   Message("[@*USER]The DistributeDocumentPack Exit Point has not yet been
 implemented");
```

Unzip

CCM Core decompresses the zip file `File` to the folder `Folder`.

Syntax

```
Unzip
   File(<text>)
   Folder(<text>);
```

Parameters

- `File`: Required. The zip file CCM Core extracts files from.
- `Folder`: Required. The folder that CCM Core extracts the contents of the zip file to.

Remarks

- The `Unzip` command extracts all files and folders in `File` recursively.
- The folder must exist; otherwise, the `Unzip` command fails.
- Files inside the folder are overwritten by files in the zip file if the file names are identical.
- The `Unzip` command cannot handle files larger than 2GB.

Wait

CCM Core waits until the specified file is created and unlocked by other applications.

Syntax

```
Wait
   File(<text>)
   TimeOut(<number>);
```

Parameters

- `File`: Required. The file for which CCM Core waits.
- `TimeOut`: Optional. The maximum amount of time in seconds that CCM Core waits if the file does not exist or if another process is locking the file. If this parameter is not specified, CCM Core waits indefinitely for the file to be created or unlocked. If the value of this parameter is 0 and `File` is locked, CCM Core immediately reports an error.

WriteFile

CCM Core appends a single line to a file. CCM Core creates the file if it does not exist. CCM Core terminates the line by a CR/LF pair.

Syntax

```
WriteFile
   File(<text>)
   Message(<text>)
   Unicode(<True or False>)
   ByteOrderMark(<True or False>);
```

Parameters

- `File`: Required. The file to which the line is appended.
- `Message`: Required. The line appended to the file.
- `Unicode`: Optional. Indicates whether the line is written in Unicode or in the local code page.

  If this parameter is not present or its value is False, CCM Core writes the line in the single-byte code page of the system. If this parameter is True, CCM Core writes the line as double-byte Unicode characters.
- `ByteOrderMark`: Optional. Indicates whether a byte order mark (BOM) should be written at the beginning of a new file. This parameter has no effect for non-Unicode encodings. It also has no effect if the target file already exists.

  If this parameter is not present or its value is False, CCM Core does not write a byte order mark. If this parameter is True, and the target file does not yet exist, CCM Core writes a byte order mark at the start of the file to indicate the Unicode encoding of the file.

Zip

CCM Core compresses the contents of folder `Folder` into the zip file `File`.

Syntax

```
Zip
   Folder(<text>)
   File(<text>);
```

Parameters

- `Folder`: Required. The folder that CCM Core compresses.
- `File`: Required. The zip file that CCM Core creates.

Remarks

The `Zip` command compresses all files and folders within `Folder` recursively.

An existing zip file `File` is silently overwritten.

The `Zip` command cannot handle files larger than 2 GB.

Chapter 4

Functions

CCM Core supports a set of built-in functions that can you can use wherever an expression is expected. Functions receive a list of comma-separated parameters as input and return a single value as their result. The parameters are enclosed in braces.

The names of the functions are case-sensitive.

directory_exists (x)

The `directory_exists(x)` function returns `True` if directory `x` exists; `False` otherwise. `x` must be a fully qualified path.

If `x` contains ? or * wildcards, the `directory_exists (x)` function returns `True` if at least one directory matches the wildcard pattern. If any component in the path does not exist or is inaccessible, the function returns `False`.

This function returns a value of type Boolean.

Example

```
directory_exists ("c:\somedirectory")
```

Result: `True` if the directory `c:\somedirectory` exists.

`directory_exists` returns `False` when used on the root of either of the following:

- Shares (`\\host\share`)
- Drives (`c:\`)

document_metadata (k)

The `document_metadata(k)` function retrieves the document metadata that has been set by the last template for the key `k`. Document metadata can be set by assigning values to fields in the built-in _Document Field Set.

If no template has previously been run or if the last template did not set a value for the key `k`, the `document_metadata(k)` function returns an empty string.

Note The `document_metadata` data is part of the session data and persists over jobs until the next template is run.

exchange_data (k, v, t)

The `exchange_data(k, v, t)` function is used to exchange data between a CCM Core server and a client.

CCM Core sends the value `v` to the client and waits for the client to send a response back.

Identifier `k` can be used by the client to identify the data.

Identifier `t` is the amount of time that CCM Core waits for the client to send its response back. After this time has elapsed without an answer, CCM Core closes the connection and reports an error. The client can handle this error but is not able to communicate with the server. CCM Core does not time out if this parameter is set to 0.

The following applies to this function:

- The function requires a synchronous connection.
- The function returns the response from the client.
- If the function causes an error, the return value is unspecified.
- If this function is used and the client does not respond within the timeout period, the Document Processor handling the job disconnects from the CCM Core Processor Manager and reconnects after the job has finished. This disconnect is necessary to resynchronize communication.

You can also use the function `exchange_data` in combination with the web services interface to allow for communication between the caller and the CCM Core script. For more information, see "SubmitEx" in the *Kofax Customer Communications Manager Core Developer's Guide*.

file_exists (x)

The `file_exists(x)` function returns `True` if file `x` exists; `False` otherwise. `x` must be a fully qualified file name.

If the file part of x contains ? or * wildcards, the `file_exists(x)` function returns `True` if there is at least one file in the directory that matches the wildcard pattern. If any component in the path does not exist or is inaccessible, the function returns `False`.

This function returns a value of type Boolean.

Example

```
file_exists ("c:\mydir\fake.doc")
```

Result: `True` if the file `fake.doc` exists in the directory `c:\mydir`.

file_format (x)

The `file_format(x)` function returns the type of file `x`. The file must exist.

Possible return values:
- `unknown`: Unable to determine the type of the file
- `doc`: Microsoft Word document
- `docx`: Microsoft Word Open XML document
- `rtf`: Rich Text Format (RTF) document
- `ps`: PostScript file
- `pdf`: PDF document
- `sxw`: OpenOffice.org / StarOffice
- `odt`: Open Document Format (ODF)
- `lwp`: WordPro document
- `wpd`: WordPerfect document
- `html`: HTML document

These are values of type Text.

Example

```
file_format ("c:\mydir\somefile")
```

Result: one of the listed responses.

get_cm_setting (x)

The `get_cm_setting (x)` functions returns the configured value for setting `x` in the Contract Manager.

The following settings can be retrieved:
- RepositoryHost
- RepositoryPort
- OutputManagementHotfolder

The function returns a value of type Text.

Note You can only use this function with Contract Manager. For information on how to develop scripts that work in Contract Manager, see the chapter "Work with the Contract Manager" in the *Kofax Customer Communications Manager Getting Started Guide*.

get_ini_setting (f, s, k)

The `get_ini_setting(f, s, k)` function returns the value of the key `k` in the section `s` of the initialization (ini) file `f`. All three parameters are of type Text. If the file `f` does not exist, or the section `s` cannot be located in this file, or the key `k` cannot be found in this section, the empty string is returned.

This function returns a value of type Text.

Example

If the file "dp.ini" contains the following section:

```
[Configuration]
ItpServerName = core_01_5.2
```

Then

```
 get_ini_setting ("dp.ini", "Configuration", "ItpServerName")
```

results in: `"core_01_5.2"`

get_sessionparameter (p)

The `get_sessionparameter(p)` function returns the value of session parameter `x` if it has a value; otherwise, the functions return an empty string.

The session parameter `x` must previously have been set in a script associated with the same session that the current script is associated with, either by means of the command `SetSessionParameter` or by means of the function set_`sessionparameter`.

For more information on sessions, see CCM Core sessions.

This function returns a value of type Text.

Examples

```
SetSessionParameter
  Par ("plugh")
  Value ("plover");
Var Text X = get_sessionparameter ("plugh");
Var Text Y = get_sessionparameter ("notplugh");
```

Result: `"plover"` and `""`

Remarks

This function throws an error when the current script is not associated with a session. If CCM Core has been instructed to ignore errors using the command `OnError Script(*)`, and an error occurs, the function returns `""`, and the _error constant is set to True.

Note Session parameters with names that begin with _itp or ITP are reserved for internal use in CCM products.

set_sessionparameter (p, v)

The `set_sessionparameter(p, v)` function assigns the value `v` to the session parameter `p`. You can use session parameters to store values between jobs associated with the same session.

For more information, see CCM Core sessions.

This function returns a value of type Boolean, indicating success or failure.

Example

```
Var Boolean Dummy = set_sessionparameter ("plugh", "plover");
Var Text X = get_sessionparameter ("plugh");
```

Result: `"plover"`

Remarks

If the function succeeds, the return value is True. If the current script is not associated with a session, or if the function fails in some other way, an error is thrown. If CCM Core has been instructed to ignore errors using the command `OnError Script(*)`, and an error occurs, the function returns False, and the _error constant is set to True.

To avoid the use of a dummy variable, use the command `SetSessionParameter` instead.

```
SetSessionParameter
  Par("plugh")
  Value("plover");
Var Text X = get_sessionparameter ("plugh");
```

Result: `"plover"`

Note Session parameters with names starting with _itp or ITP are reserved for internal use in CCM products.

get_document_from_pack (s, c, t, d)

The `get_document_from_pack (s, c, t, d)` function returns the fully qualified path to a document in the active Document Pack. The pack is selected using the following attributes:

`s` is the name of the slot in the Document Pack.

`c` is the name of the channel. `""` indicates the default channel. If the channel is not present and `d` is True, the document in the default channel is returned instead.

`t` is the type of the document that should be selected. `""` selects the original output of the template, regardless of its type.

`d` indicates, when `c` indicates a specific channel, that if the channel is not present, the default channel should be selected instead.

If the document cannot be found or there is no active Document Pack in the channel, the function returns an empty string. If the Document Pack explicitly contains no document for the channel, the function returns `"*none"`.

Examples

```
get_document_from_pack ("Coverletter", "", "", False)
```

returns the original (DOC, DOCX or HTML) result document produced by the template in the `Coverletter` slot.

```
 get_document_from_pack ("Coverletter", "", "pdf", False)
```

returns the PDF alternative for the `Coverletter` slot.

```
get_document_from_pack ("Coverletter", "email", "_xml", True)
```

returns the AiaDocXML alternative for the `Coverletter` slot for the email channel. If this channel is not present in the pack, the alternative for the default channel is returned instead.

get_slots_from_pack (c)

The `get_slots_from_pack (c)` function returns a list of slots in the active Document Pack that define a document for a channel.

`c` is the name of the channel. `""` indicates the default channel and returns a list of all slots in the Document Pack.

The function returns a string that contains all matching slots, which are separated by tabs. If there is no active Document Pack, the function returns an empty string.

Examples

```
get_slots_from_pack ("Archive")
```

returns a list of slots that define a document for the `Archive` channel.

```
Var Text slot;
ForEach slot
  In get_slots_from_pack ("")
  Separator _Tab
Do
  Progress Message ("Slot: " + slot);
Od;
```

sends a progress message for each slot in the Document Pack.

get_channels_from_pack (s)

The `get_channels_from_pack (s)` function returns a list of channels in the active Document that are defined for a slot.

`s` is the name of the slot. `""` lists all channels defined for all slots.

The function returns a string that contains all matching channels, which are separated by tabs. If there is no active Document Pack, the function returns an empty string.

Examples

```
get_channels_from_pack("Coverletter")
```

returns a list of channels defined for the `Coverletter` slot.

```
Var Text channel;
ForEach channel
  In get_channels_from_pack ("")
```

```
  Separator _Tab
Do
  Progress Message ("Channel: " + channel);
Od;
```

sends a progress message for each channel in the Document Pack.

hex (x)

The `hex(x)` function converts the number `x` to a hexadecimal representation.

This function returns a value of type Text.

Example

```
 hex (15)
```

Result: `"f"`

index (x, s)

The `index(x, s)` function searches for the first occurrence of the string `s` in `x`.

It returns the index of the first character of the occurrence or 0 if the string `s` did not occur in `x`.

The search is case-insensitive.

This function returns a value of type Number.

Examples

```
 index ("abcdabcd", "cd");
```

Result: `3`.

```
 index ("AbCdAbcd", "cd");
```

Result: `3`. Note that the search is case-insensitive.

rindex (x, s)

The `rindex(x,s)` function searches for the last occurrence of the string `s` in `x`.

It returns the offset of the first character of the last occurrence or 0 if the string `s` did not occur in `x`.

The search is case-insensitive.

This function returns a value of type Number.

Examples

```
rindex ("abcdabcd", "cd")
```

Result: `7`.

```
rindex ("AbCdaBcD", "cd")
```

Result: `7`. Note that the search is case-insensitive.

itp_parameter (k)

This function is deprecated. Use the `document_metadata` function instead. For more information, see document_metadata (k).

The `itp_parameter (k)` function retrieves the value that has been set by the last template for the key `k`. The template sets this value by calling the function `itpserver_parameter ("k"; "value")`.

If no template has previously been run by the job or if the last template did not set a value for the key `k`, the `itp_parameter (k)` function returns an empty string.

Use the `itp_parameter(...)` function in combination with the `itpserver_parameter(…)` function to pass data from a template to the calling CCM Core script.

The `itp_parameter` data is part of the session data and persists over jobs until the next template is run. You can use this function to retrieve information from the template. The template can use `runmodel_setting`, `itpserver_setting`, and `environment_setting` to retrieve information from its context. See the sections runmodel_setting, itpserver_setting, and environment_setting in the *Kofax Customer Communications Manager Template Scripting Language Developer's Guide*.

length (x)

The `length(x)` function returns the number of characters in the parameter `x`.

This function returns a value of type Number.

Examples

```
length (a_string_or_a_text_variable)
```

Result: The number of characters in the content of the variable `a_string_or_a_text_variable`.

```
length ("abcd")
```

Result: `4`.

number (x)

The `number(x)` function converts the string `x` to an integral number. The input should not be formatted.

This function returns a value of type Number.

Example

```
number ("42")
```

Result: `42`.

text (x)

The `text(x)` function converts the number `x` to a textual representation.

This function returns a value of type Text.

Example

```
 text (4)
```

Result: `"4"`

replace (x, s, t)

The `replace(x, s, t)` function returns the content of the Text parameter `x`, where all occurrences of the Text parameter `s` are replaced by the content of the Text parameter `t`. If `s` is empty, nothing is replaced, and the result is equal to the original parameter `x`. The search and replace is done from left to right, and neither overlapping substrings nor already replaced text are considered for replacement.

The replace function is case-sensitive.

This function returns a value of type Text.

Examples

```
 replace ("The Quick Brown Fox Jumps Over The
Lazy Dog", "The", "a")
```

Result: `"a Quick Brown Fox Jumps Over a Lazy Dog"`

```
 replace ("The Quick Brown Fox Jumps Over The
Lazy Dog", "the", "that")
```

Result: `"The Quick Brown Fox Jumps Over The Lazy Dog"`

```
replace ("The Quick Brown Fox Jumps OOOver The
Lazy Dog", "OO", "oOOo")
```

Result: `"The Quick Brown Fox Jumps oOOoOver The Lazy Dog"`

rsubstring (x, s, n)

The `rsubstring(x, s, n)` function returns a substring from the parameter `x`. The substring includes the `s` character of `x` and all but the last `n` characters of `x`.

The behavior of `rsubstring` can be expressed in the following way.

```
rsubstring(x, s, n) = substring(x, s,((length (x) - s) - n))
```

If `s` is negative, the substring is started at the `s` character from the end of the string and counts towards the end.

If `n` is negative, the substring contains all characters from the `s` character up to the end of the string.

This function returns a value of type Text.

Common use

```
remainder = rsubstring(x, s, 0)
```

This returns the remainder of a string starting at position s in the string. The third parameter `0` indicates that no characters from the end of the string are omitted.

```
remainder = rsubstring(x, 0, 4)
```

This returns the input string without the last 4 characters.

Examples

```
rsubstring ("This is a counting test", 0, 3)
```

Result: `"This is a counting t"`. `0` sets the start position at the beginning of the input string and `3` indicates that the last three characters from the end of the input string must be left out.

```
rsubstring ("This is a counting test", 4, 0)
```

Result: `"s is a counting test"`. `4` sets the start position in the input string and `0` indicates that no characters must be left out.

```
rsubstring ("This is a counting test", -8, 3)
```

Result: `"ing t"`. `-8` selects the last 8 characters from the end of the string (`"ing test"`) and `3` indicates that all but the last 3 characters of that substring are returned.

```
rsubstring ("This is a counting test", 3, 25)
```

Result: `""`, an empty string. The third parameter indicates that 25 characters from the end of the input string must be left out. As there are less than 25 characters in the input string from the start position (parameter two: 3) in the input string, the result is an empty string.

substring (x, s, n)

The `substring(x, s, n)` function returns a substring from the parameter `x`. The substring includes the `s`th character of `x` and contains `n` characters.

If `x` does not contain sufficient characters, the result is truncated.

If `s` is negative, the substring is started at the `s` character from the end of the string and counts towards the end.

If `n` is zero or negative, the substring `w` returns an empty string.

This function returns a value of type Text.

Examples

```
substring ("This is a counting test", 4, 9)
```

Result: `"s is a co"`

```
substring ("This is a counting test", 4, 0)
```

Result: `""`, an empty string.

```
substring ("This is a counting test", 4, -5)
```

Result: `""`, an empty string.

```
substring ("This is a counting test", -4, 9)
```

Result: `"test"` The start position is taken from the end of the string because -4 is negative. From that position the string only contains 4 more characters, which are returned as the result.

```
substring ("This is a test in counting", 4, 25)
```

Result: `"s is a test in counting"` From the starting position the string only contains 23 more characters, which are returned as the result.

template_property (k)

The `template_property(k)` function retrieves the template property that has been set by the last template for the key `k`. Template properties are set on Document Templates in CCM Designer. Template properties are available to the template as Fields in the _Template Field Set. These values are read-only.

If no template has previously been run or if the last template did not set a value for the key `k`, the `template_property(k)` function returns an empty string.

Note The `template_property` data is part of the session data and persists over jobs until the next template is run.

toupper (s)

The `toupper(s)` function returns the string `s` in which all lowercase characters are converted to their uppercase equivalents. All other characters are passed unmodified.

This function returns a value of type Text.

Example

```
toupper ("New York")
```

Result: `"NEW YORK"`

tolower (s)

The `tolower(s)` function returns the string `s` in which all uppercase characters are converted to their lowercase equivalents. All other characters are passed unmodified.

This function returns a value of type Text.

Example

```
tolower ("New York")
```

Result: `"new york"`

Chapter 5

Conditional statements and iterations

You can use conditional statements and iterations to affect the flow of control through a CCM Core script.

Conditional and iterating statements also limit the scope of variables and constants declared within them. You can only use a variable or constant declared in a statement within that statement scope.

If...Then...Fi

You can use the `If` statement to conditionally execute a block of statements.

Syntax

```
If bool_expression Then
 code;
Elif bool_expression Then
 code;
Elif bool_expression Then
  code;
Else
  code;
Fi;
```

`If bool_expression Then` is the first expression.

`code;` is executed if the first expression is True.

`Elif bool_expression Then` is the second expression.

`code;` is executed if the first expression is False and the second is True.

`Else` is executed if none of the preceding expressions is True.

The instruction `If … Then … Fi` can contain zero, one or more `Elif … Then … parts`, which specify alternative conditions selected if none of the preceding conditions evaluated to True.

The instruction `If … Then … Fi` can optionally end with an `Else …` statement, which is executed if none of the preceding conditions evaluated to True.

Note the closing semicolon of each expression and the closing semicolon after the `Fi`.

CCM Core first evaluates the `bool_expression` between the `If` and `Then` statements:

- If this expression evaluates to True, CCM Core executes the instructions following the `Then` statement up to the next `Elif`, `Else`, or `Fi` and then continues executing instructions after `Fi`.
- If this expression evaluates to False, CCM Core skips the following instructions and continues at the next `Elif`, `Else`, or `Fi`. If the next instruction is an `Elif` statement, CCM Core repeats this process

by evaluating the expression between the `Elif` and `Then` statements. If the next instruction is an `Else` statement, CCM Core executes the instructions between `Else` and `Fi`.

The conditional blocks of statements can contain any valid CCM Core instructions including other conditional statements. The only exception is the Parameter statement, which you can only use at the beginning of the script.

For...=...To...Step...Do...Od

The `For` statement is used to repeat a loop based on a counter.

Syntax

```
For variable = number
  To number
Step number
Do
  ... ;
Od;
```

`For variable = number` is the counter and initial value.

`To number` is upper limit.

`Step number` is optionally step size.

`... ;` is repeated code.

Variable must be declared as a Number. The initial value, limit, and step values are calculated once before the loop is executed the first time. At the end of each loop variable is incremented with the `Step` value. It is possible (but not advisable) to modify the variable within the loop.

If the `Step` keyword is omitted, the step value defaults to 1.

The `Do … Od` loop is executed while:

- Variable is smaller or equal to the `To` value if the `Step` value is greater or equal to 0.
- Variable is larger or equal to the `To` value if the `Step` value is negative.

You can use the `Break` command to terminate the loop unconditionally.

Example

This example locates a slash in a path by walking through it using the substring function.

```
Parameter Text Path;
Var Number Count;
For Count = 1 To length(Path) Do
  If substring (Path, Count, 1) = "\" Then
    Break;
  Fi;
Od;
```

`Parameter Text Path;` The parameter `Path` is the input passed to this script.

`Var Number Count;` The count variable needed in the statement `For`. Must be declared before this statement.

The code `If substring (Path, Count, 1) = "\" Then Break;` between `Do` and `OD` will be repeated until either the function substring (`Path, Count, 1`) is equal to `"\"` or `Count` is larger than length (`Path`). When `substring (Path, Count, 1)` is equal to `"\"`, the loop is terminated. Count at that point will contain the position of `"\"` in `Path`. You can use `Count` in the remainder of the script, because `Count` is declared outside the `For` loop which means that its scope is not limited to the `For` loop. Although this is a valid example, the file name manipulation methods provided in CCM Core are more convenient.

ForEach...In...Do...Od

You can use the statement `ForEach … In` to process elements in a comma-separated list.

Syntax

```
ForEach variable
In comma_separated_list
Do
  ... ;
Od;

ForEach variable
In list
Separator sep
Do
  ... ;
Od;
```

`ForEach variable` is the variable declared as Text. Common practice is to use the name `Element` for this variable to enhance readability.

`In comma_separated_list` is a list declared as a Text variable.

Example

```
Var Text List="1,2,3"; Variable Text List = "1,2,3,..."
```

`Separator sep` (optional) is the text used to separate elements in the list. If omitted or when `sep` is an empty text, the comma is used as default separator.

`... ;` is the code to be repeated for each element in the list.

The list is evaluated as a list of values where leading and trailing spaces are stripped.

Every element is assigned to the variable before the `Do … Od` code is executed.

Empty elements in the list are ignored.

Example

```
Var Text List = "1,2,4"
```

The third empty element is ignored. Only for the elements `1`, `2`, and `4` the code between `Do ... Od` is executed.

You can use the command `Break` to terminate the loop unconditionally.

If the `List` is empty, an error is not triggered, and the code between the `Do ... Od` part of the statement is not executed.

Example

```
Var Text Element;
Var Text List = "one, 2, yesterday";

ForEach Element
In List
Do
  Progress Message (Element);
Od;
```

This script sends the progress messages `one`, `2`, and `yesterday` back to the client.

Note that in effect the variable `Element` is assigned the next value from `List` at each passing. As a result, the messages in the preceding script are `one`, `2`, and `yesterday`.

The keyword `Separator` allows the script to split the list on other strings.

Example

```
Var Text Element;
Var Text List = "one|+ 2|+ yesterday";

ForEach Element
In List
Separator "|+"
Do
  Progress Message (Element);
Od;
```

You can use the built-in variable _newline to split multiline output from external commands into the separate lines and process those lines separately.

```
Var Text Line;

RunCommand CmdLine (...);

ForEach Line
  In _stdout
  Separator _newline
Do
  /* process the line */
  ...
Od;
```

ForEach...File...Do...Od

You can use the `ForEach … File` statement to process lines from a file.

Syntax

```
ForEach Line
encoding File "path\filename"
```

```
Do
... ;
Od;
```

`ForEach Line` is a variable declared as Text. Common practice is to use `Line` as variable name because it contains the current line at each passing.

`File "path\filename"` is the file path and name. Each line of this file is used to execute the part `Do ... Od`

`... ;` is repeated code.

Optionally, the script can specify an encoding. Supported encoding keywords are:

- `UTF8`
  The data is interpreted as variable-length UTF-8 encoding.
  Invalid byte sequences are mapped to the U+FFFD REPLACEMENT CHARACTER (_invalidchar).
- `UTF16`
  The data is interpreted as double-byte little-endian (Windows) UTF-16 characters.
- `Auto`
  If the file begins with an UTF-8 or UTF-16 BOM (byte order mark), the file is read in the appropriate encoding. If no BOM is found, the file is interpreted as single-byte characters in the system ANSI codepage.

If no encoding is specified, the file is interpreted as single-byte characters in the system ANSI codepage.

Every line is assigned to the variable before `Do … Od` is executed. You can use the `Break` command to terminate the loop unconditionally.

Example

```
Var Text Line;

ForEach UTF8 Line
File "c:\temp\log"
Do
  Progress Message (Line);
Od;
```

This reads the file `C:\temp\log`, interprets the data as UTF-8, and sends every line back to the client as a separate message.

ForEach...Folder...Do...Od

You can use the statement `ForEach ... Folder` to process objects in a folder.

Syntax

```
Var Text File;
ForEach File
Folder "path\foldername"
Do
  ... ;
Od;
```

The `Foreach ... Folder` loops over all files and directories in a folder. For each object, the name is assigned to the variable `File` before the `Do ... Od` code is processed.

You can use the command `Break` to terminate the loop prematurely.

Note Any directories in the folder are also processed. Code within the loop should detect or handle directories.

The content of directories is not processed. Recursion must be handled explicitly by the script.

If the contents of the folder change while `ForEach ... Folder` is processing, the behaviour is undefined.

Example

```
Var Text File;
Var Text ResultDirectory = "C:\ITPResult";

ForEach File
Folder ResultDirectory
Do
 If file_format(File) = "doc"
  THEN
   DoctoPDF
    Src (File)
    Dest (File[,"pdf"]);
 Fi;
Od;
```

This converts all Microsoft Word files in the `c:\ITPResult` directory to PDF.

While

You can use the statement `While` to repeat a loop based on a condition.

Syntax

```
 While <Boolean expression>
Do
...;
Od;
```

The condition is evaluated before every iteration of the loop. If the condition evaluates to True, the loop is executed. If the condition evaluates to False, CCM Core continues with the next statement following `Od`.

You can use the `Break` command to terminate the loop unconditionally.

Example

```
Parameter Text String;

While String <> ""
Do
  Progress Message (substring (String, 1, 1));
  String = rsubstring (String, 2, 0);
Od;
```

This script sends all characters from the parameter to the client, one character at a time. Note the use of `rsubstring`. This function returns the input string minus the first character: the `2` parameter takes care of that, and the `0` parameter says that no characters must be left out from the end of the input string.

Break

The statement `Break` ends `For … Do … Od`, `ForEach … Do … Od`, and `While … Do … Od` loops. This statement is only allowed in repeating controls. `Break` is a command in itself and must be followed by a semicolon.

Example

```
Parameter Text Path;
Var Number Count;

For Count = 1 To length(Path) Do
  If substring (Path, Count, 1) = "\" Then
    Break;
  Fi;
Od;
```

This code searches for the first slash in the parameter and returns its position.

Return

The statement `Return` ends a script.

if a script containing a return statement is called from another script, the calling script is not ended by that return statement.

Example

```
Parameter Text Document = "";

If Document = "" Then
  Return;
Else
  PrintDocument
    Src (Document);
Fi;
```

Result: If the `If` statements evaluates to True, the return statement ends this script.

Chapter 6

External tools

You can use a Visual Basic script SetReadOnly to set the Read Only attribute of any file as part of the CCM Core distribution.

Read the header of the script SetReadOnly carefully before use. For more information, see SetReadOnly.vbs.

ITPWinMon

The functionality of the ITPWinMon program is integrated into CCM Document Processor. To avoid interference, the program ITPWinMon has been replaced with a dummy program. You can remove the startup of the program ITPWinMon in the UserStartUp.dss script.

SetReadOnly.vbs

This script, which sets the read-only attribute of a file, requires three parameters:

It requires three parameters:

1. The full path to the file.
2. The required Read Only state: y or Y makes the file read only, and all other characters allow users to edit the file.
3. The full path to a log file, which is created by the script if an error occurs. In this case, an error message is written to the file (if possible). If the special value *DEBUG is provided as the third parameter, error messages are shown interactively.

The script returns one of the following codes:

- 0: the script was successful
- 1: the script failed; an error message has been written to the log file
- 2: the script failed and was unable to write an error message to the log file
- 3: the script was not called according to its signature (wrong number of parameters)

Note SetReadOnly.vbs needs the Microsoft Scripting Runtime to be activated on the machine running CCM Core.

This example script demonstrates how `SetReadOnly` can be called from within a CCM Core script.

```
# ITP/Server script SetReadOnly
```

```
Parameter Text File;
Parameter Boolean ReadOnly;
Parameter Text LogFile;

Const Text pathtovbs = "C:\Program files\ITP Server\Tools\SetReadOnly.vbs";
# The declaration above is one line!

Var Text readonlychar = "N";
If (ReadOnly) Then
 readonlychar = "Y";
Fi;

RunCommand
  CmdLine("WScript.exe //B //nologo " +
          """ + pathtovbs + """ " +
          """ + File + """ " +
          """ + readonlychar + """ " +
          """ + LogFile + """");
# Please note that all quotes in CMDLine are needed. (quotes are
# used to escape quotes)
```

ChangeBins tool

Microsoft Word allows users to configure paper trays for the first and subsequent pages of every section in a document. The selection of available paper trays is always limited to the trays offered by the currently selected printer.

If the user selects another printer to print a document on, the user has to change the selected paper tray unless the new printer is the same brand and model as the original printer.

Microsoft Word stores the selected paper trays by "bin number." These bin numbers are defined by the printer driver and can vary from printer to printer, depending on the options installed on that printer. If another printer is selected, Word simply matches paper trays based on the bin numbers and maps all bins unknown by the new printer to its default paper tray. The names of the bins and their definitions are ignored, and it is left to the user to correct any mismatches.

If an organization uses different brands or models of printers, this behavior can lead to unpredictable results:

- If the new printer driver uses the same bin number as the printer driver the document was formatted for, the pages are printed from that paper tray regardless of the properties of the tray.
- If the new printer driver does not use the same bin number as the printer driver the document was formatted for, the pages are printed from the default tray.

The ChangeBins tools have been developed to solve the problems described earlier. These tools allow an organization to specify a mapping for the printer drivers, which you can use to change paper trays according to the real attributes of the printer trays.

The following programs are included in the ChangeBins tools:

- ChangeBins
  A command-line version designed to develop and test printer configurations.
- ChangeBinsW

A Windows version of the ChangeBins program designed to be used in an unattended environment. This program only provides a return code and does not print additional diagnostics or use message boxes.

- QueryPrinters
  A command-line tool designed to query bin number assignments for all printers installed on a computer.

ChangeBins configuration

Identify paper trays

The ChangeBins tool requires a mapping of printer tray numbers for all applicable printer types. The QueryPrinters tool examines all printer drivers installed on a system and reports the paper trays and their bin numbers. You can use these bin numbers to construct a mapping.

Sample output from the QueryPrinters tool.

```
[01] PS_LJ4000                                  Name of the printer as shown in Windows
     Port (list): LPT1:
     Driver: AdobePS HP LaserJet4000SeriesPS
     Printer provides 7 bins
     [01] (015) Automatically Select
     [02] (257)  Tray 1                         (tray number) and name of the tray as shown in Word
     [03] (258)  Tray 1 (Manual)
     [04] (259)  Tray 2
     [05] (260)  Tray 3
     [06] (261)  Tray 4
     [07] (262)  Envelope Feeder

[02] HP LaserJet 4050 Series PS
     Port (list): LPT2:
     Driver: AdobePS HP LaserJet4050SeriesPS
     Printer provides 24 bins
     [01] (015) Automatically Select
     [02] (257)  Tray 1
     [03] (258)  Tray 1 (Manual)
     [04] (259)  Tray 2
     [05] (260)  Tray 3
     [06] (261)  Tray 4
     [07] (262)  Tray 5
     [08] (263)  Tray 6
     [09] (264)  Tray 7
     [10] (265)  Tray 8
     [11] (266)  Tray 9
     [12] (267)  Tray 10
     [13] (268)  Envelope Feeder
     [14] (269) Plain                           The HP LaserJet 4050 can match paper types
     [15] (270) Preprinted                      with a paper tray based on their type, which
     [16] (271) Letterhead                      allows operators to load paper in any tray as
     [17] (272) Transparency                    required.
     [18] (273) Prepunched                      This is shown as virtual "paper trays" in Windows.
     [19] (274) Labels
     [20] (275) Bond
     [21] (276) Recycled
     [22] (277) Color
     [23] (278) Card Stock
     [24] (279) Rough
```

```
[03] \\xerox\xerox                          Network printers show up with their path.
     Port (list): \\xerox\xerox
     Driver: HP LaserJet 5P
     Printer provides 5 bins
     [01] (015) Automatically Select
     [02] (257) First Available Tray
     [03] (001) Tray 1
     [04] (004) Manual Paper Feed
     [05] (002) Tray 2
```

This example shows three printers with their paper trays (bins) and the bin numbers assigned to those trays. Note that different assignments are used for common paper trays by the drivers described in the table.

Printer	Tray 1	Tray 2	Envelope Feed
PS_LJ4000	257	259	262
HP LaserJet 4050 Series PS	257	259	268
\\xerox\xerox	1	2	-

Map input trays

Based on the identified trays, you can construct a mapping. This mapping must be based on the types of paper used.

A sample configuration is described in the table.

Printer	Paper Types			Remarks
	Plain	*Letterhead*	*Pre-printed*	
PS_LJ4000	Tray 4 (261)	Tray 1 (257)	Tray 2 (259)	Trays are loaded with a fixed paper type.
HP LaserJet 4050 Series PS	Plain (269)	Letterhead (271)	Preprinted (270)	Operators load available trays as demanded.
\\xerox\xerox	Tray 2 (2)	N/A	N/A	This printer has only plain paper loaded.

Create the configuration file

The ChangeBins and ChangeBinsW programs read their printer mappings from the configuration files ChangeBins.cfg and ChangeBinsW.cfg, respectively. These files must be present in the same directory as the programs.

The configuration files are text files and you can edit them with any text editor. Unicode and UTF-8 encodings are not supported.

For every printer configuration, a line must be present that contains the following information:

- The name of the printer in quotation marks
- One or more paper tray numbers, separated by spaces or tabs

All configured printers must have the same number of paper trays in their specification. The trays are matched based on their order of occurrence. If a printer does not provide a paper tray, specify an alternative. It is allowed to specify the same paper tray more than once.

Whenever a printer mapping is performed, the ChangeBins tools look up the configuration for the printer used to configure the Microsoft Word document. For every paper selection used in this document, it looks up the first occurrence of this tray number in the configuration. If the number occurs, it is replaced with the matching tray number from the configuration of the target printer. If the tray number does not occur in the configuration, it is replaced by "0" (no tray selected). This instructs Microsoft Word to fall back to the Auto Select tray when it prints the document.

A sample configuration:

```
"PS_LJ4000"                  0  15  261  257  259
"HP LaserJet 4050 Series PS" 0  15  269  271  270
"\\xerox\xerox"              0  15  2    2    2
```

In this example, the paper trays are specified in the following order.

1. Bin 0 (defaults to AutoSelect for DOC documents)
2. In this example, `15` indicates auto select for the various printers
3. Plain paper
4. Letterhead paper
5. Preprinted paper

The `\\xerox\xerox` printer in this example has only a single paper tray, so all paper types are mapped to Plain paper as replacements.

It is not necessary to provide mappings for every available printer. If printers are equipped with identical options, their configurations should also be identical, and it is sufficient to provide a generic printer identification to match all devices.

Map paper trays

The CCM Core scripting language provides the ChangeBins script command that automates the call to the ChangeBins program. The following information is provided for situations where the ChangeBins tools have to be invoked directly.

To change the paper trays in a Microsoft Word document, the ChangeBins tools must know which printer has been used to make the Microsoft Word document and on which printer the document has to be printed. This information must be issued on the command line.

```
ChangeBins "document.doc" "original printer" "new printer"
```

or

```
ChangeBinsW "document.doc" "original printer" "new printer"
```

where:

`document.doc` is the full path to the document that must be changed.

`original printer` is the printer used to format the document.

`new printer` is the printer used to print the document.

The quotation marks are required to ensure that spaces are handled correctly.

The ChangeBins and ChangeBinsW programs also accept one or more `/Section` options to provide more detailed paper tray handling. This option is described in Selective tray changes.

To change the document `"C:\My Documents\sample.doc"` formatted for a `"HP LaserJet 4050 Series PS"` printer to print on the `"\\xerox\xerox"` printer, use the following command line (line breaks have been added for readability).

```
 "C:\Program Files\ChangeBins\ChangeBins"
"C:\My Documents\sample.doc"
"HP LaserJet 4050 Series PS"
"\\xerox\xerox"
```

The ChangeBins program shows the changes it made to the sample document.

```
Changing bins from "HP LaserJet 4050 Series PS" to "\\xerox\xerox"
Changing Bins of Section 1:
        First Page: Changed Bin 271 to Bin 2
        Following Pages: Changed Bin 270 to Bin 2
Changing Bins of Section 2:
        First Page: Changed Bin 269 to Bin 2
        Following Pages: Changed Bin 269 to Bin 2
```

Return codes

Both ChangeBins and ChangeBinsW provide error information through their return codes:

- 0 Completed successfully
- 1 An error occurred

If an error occurs, the ChangeBins program writes a diagnostic error to its output. This information shows when the ChangeBins program is called from Command Prompt window.

Selective tray changes

You can also use the ChangeBins tool to change tray settings with command line parameters. These parameters allow a more detailed configuration of the affected sections.

`/Section` **parameter**

You can call the ChangeBins and ChangeBinsW programs with zero, one, or more `/Section` options. If the ChangeBins script command is used, the `/Section` options can be provided through the `Overrides` parameter.

Each one of these options has the following format.

```
/Section <section>:<first page>[,<other pages>]
```

where:

`section` Number of the section to which the changes apply.

The section number should either be the number of the section, or `*` to affect all sections in the document. It is allowed to have both `/Section *:...` and specific `/Section` parameters. In this case the specific settings have priority over the general `*` setting.

`first page` Specification for the paper tray of the first page of the section. The following specifications are supported:

- `-` No change
- `n` Sets the paper tray to tray `n`.
- `?n` Looks up paper tray `n` in the configuration file and sets it to the equivalent paper tray for the new printer.
- `!` Takes the paper tray from the document and sets it to the equivalent paper tray for the new printer.

`other pages` Specification for the paper tray of the other pages of the section. The format is identical to the first page parameter. If this parameter is omitted, the first page parameter is applied to all pages in the section.

If the `/Section` option is used to apply fixed paper tray numbers, the original printer and new printer parameters of the ChangeBins and ChangeBinsW programs can be omitted. If the specification indicates a mapping and no original printer and new printer are specified, the paper tray is set to auto select.

Examples

The following examples change the paper trays of document `"test.doc"` from printer `"old"` to printer `"new"`. Examples are shown for both the ChangeBins.exe program and the ChangeBins script command. Examples for ChangeBinsW.exe have been omitted as these are identical to those of the ChangeBins.exe program.

The following is the standard behavior.

```
ChangeBins "test.doc" "old" "new"

ChangeBins
    Document ("test.doc")
    From ("old")
    To ("new");
```

or

```
ChangeBins /Section *:! "test.doc" "old" "new"

ChangeBins
    Document ("test.doc")
    From ("old")
    To ("new")
    Overrides ("/Section *:!");
```

Map tray settings

Take the paper tray settings for every section from the document and map them using the configuration file.

1. Change all sections to a single bin.

   ```
   ChangeBins /Section *:4 "test.doc"
   ```

2. Set all sections to use paper tray 4.

   ```
   ChangeBins /Section *:?4 "test.doc" "old" "new"
   ```

This maps paper tray 4 on printer "old" to the equivalent bin on printer "new" and sets all paper trays in the document to this paper tray.

3. Change the first page of the document.
   - `ChangeBins /Section 1:3,- /Section *:- "test.doc"`

     This changes the first page of the first section to paper tray 3 and leaves all other pages unmodified.
   - `ChangeBins /Section 1:3,2 /Section *:2 "test.doc"`

     This changes the first page of first section to paper tray 3 and all other pages in the document to paper tray 2.
4. Override a specific section in the document.

   ```
   ChangeBins /Section 1:3,?2 /Section *:?2 "test.doc" "old" "new"
   ```

   This changes the first page of the first section to paper tray 3 and all other pages in the document to the printer "new" equivalent of the "old" printer paper tray 2.
5. Change a specific section in the document.
   - `ChangeBins /Section 4:3,2 /Section *:- "test.doc"`

     This changes the first page of section 4 to paper tray 3 and other pages in this section to paper tray 2. Other sections of the document are not modified.
   - `ChangeBins /Section 4:3,2 "test.doc" "old" "new"`

     This changes the first page of section 4 to paper tray 3 and other pages in this section to paper tray 2. Other sections of the document are not modified.

Example of different options combined

This example performs the following changes:

- The first page of the first section is set to paper tray 300.
- Other pages in the first section are mapped from printer "old" to printer "new".
- The first page of the fifth section is set to the equivalent of paper tray 2 of printer "old".
- The other pages of the fifth section are left unmodified.
- All other pages in the document are set to paper tray 3.

```
ChangeBins /Section 1:300,! /Section 5:?2,- /Section *:3 "test.doc" "old" "new"

ChangeBins
    Document ("test.doc")
    From ("old")
    To ("new")
    Overrides ("/Section 1:300,! /Section 5:?2,- /Section *:3");
```