Kofax Customer Communications Manager

Repository Developer's Guide

Version: 5.2

Date: 2018-11-19

Table of Contents

Preface

This guide describes the use of API and automation in the ITP/MDK Microsoft Word environment. The later only applies to Kofax Customer Communications Manager Designer for Windows.

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 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.

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

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

Kofax Customer Communications Manager Core Scripting Language Developer's Guide
Describes the CCM Core Script.

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

API

This chapter provides description of the available API to CCM Repository.

Get Model API

The Get Model API is available to retrieve a Master Template (formerly known as Model) from CCM Repository without a deploy. It stores the Master Template in a folder specified by the requesting client.

Contents of the Get Model API

The GetModel.exe program additionally requires the libraries cc32230mt.dll, borlndmm.dll, aiacore.dll, and itprepbase.dll from the installation directory to be available. If you copy the GetModel.exe program to another location, make sure to also copy these libraries.

Call to the Get Model API

The syntax for calling the Get Model API client program is as follows.

```
getmodel.exe -model=<master template name> -file=<master template file>
             -cfg=<repository ini file>
             [<options>]
```

Note The code is one line.

Provide values for the parameters as follows:

- `<master template name>` Required. Specify the full path of the folder containing the Master Template in CCM Repository. If the path contains spaces, enclose the path in double quotes.

  ```
  -model=\MyProject\MyFolder\MyMasterTemplate
  ```

  or

  ```
  -model="\MyProject\MyFolder itp\some Master Template"
  ```

- `<master template file>` Required. Enter the complete path and name of the file containing the Master Template. If there are spaces, enclose the entire path in double quotes.
- `<repository ini file>` Required. Enter the complete path to the repository INI file. This file must contain the connection information to the CCM Repository server, including host name and port. If the path includes spaces, enclose the whole path in double quotes.

The following parameters are optional:

- `-r=<revision number>` Indicates the revision to be retrieved
- `-label=<label>` Indicates the revision to be retrieved.

If `-r` and `-label` are both omitted, the [draft] revision is retrieved. If the Master Template does not have a [draft] revision, CCM Repository responds with an error.

Instead of a revision number, also the text "accepted" or "published" may be passed to the `-r` option.

Example `-r=accepted`. This option retrieves the [accepted] or [published] Master Template revision respectively, if present.

Master Templates and folders that are marked for deletion are disregarded when looking for the Master Template.

Return codes

The following table lists and describes error codes that can be returned by the Get Model API.

Error code	Description
5	The specified Master Template does not exist in the CCM Repository.
6	There is no revision with the given revision number for the specified Master Template.
7	The specified label does not exist in the CCM Repository.
8	There is no revision with the given label for the specified Master Template.
9	The Master Template specified does not have a [draft] revision.
10	The Master Template specified does not have an [accepted] revision.
11	Duplicate revisions have been found for the specified Master Template and revision.
12	The Master Template specified does not have a [published] revision.
21	The server configured is not a CCM Repository server (check the CCM Repository Client configuration settings).
22	Failed to create the Master Template file.
23	Failed to write the Master Template file.
25	Call to server failed (check the CCM Repository Client log).
26	Failed to connect to the CCM Repository Server (check the CCM Repository Client configuration settings).
27	The set of parameters provided is incomplete.
28	Both a label and a revision number have been supplied.
31	Unspecified error.

Chapter 2

Text Block migration

You can use the tool TBMigrate to load text fragments from third-party applications into CCM Repository as Text Blocks or Rich Text Blocks. Text Blocks must be present in the XML format used by CCM Repository and be stored in files with the .xml extension. Rich Text Blocks must be Microsoft Word documents (either *.doc or *.docx).

Loading Microsoft Word documents as Rich Text Blocks has the following limitations:

- Fields are not supported. The content of the documents is not checked for the presence of Fields and Field Sets, and no Fields or Field Sets are created in CCM Repository as a result of loading a Rich Text Block.
- No preview version of the Rich Text Block is created. Where a preview of the imported Rich Text Block is requested, the text "Preview not available" is shown. To generate a proper preview, edit the imported Rich Text Block and save it.

Also, Rich Text Blocks have the optional extra functionality that enables you to add a file with the same name as the Rich Text Block file and the extension .nfo, which contains additional information about the Text Block. You can add characteristics to a Rich Text Block by including `characteristic:name` or `characteristic:group.name`. There should be no additional whitespace. The characteristic and group are created if they did not previously exist.

Migrate Fields and Field Sets

Fields and Field Sets are not imported explicitly but created when Fields are being referenced in the Text Blocks. This behavior can be used if you need to create large amounts of Field Sets and Fields, for example, in a migration project. For this, create a dummy Text Block XML file, containing no text but only references to the Fields and Field Sets that should be created. After importing the file with the tool TBMigrate, you can delete the dummy Text Block from CCM Repository.

Contents of the TBMigrate tool

The tool consists of the program tbmigrate.exe and a help program tbval.exe. They can be found in the "extra" folder of a CCM Designer for Windows installation, which is typically: `C:\Program Files (x86)\ITPMDK <name>\extra`. To access them, you need to install CCM Designer for Windows (see the *Kofax Customer Communications Manager Installation Guide*).

The tool is a client-side program and needs access to an itprep.ini file. This file allows the tool to connect to the CCM Repository Server. Therefore, copy both programs to the CCM Designer for Windows installation directory, which is typically: `C:\Program Files (x86)\ITPMDK <name>`.

The tbmigrate.exe program additionally needs the libraries cc32230mt.dll, borlndmm.dll, aiacore.dll, and itprepbase.dll to be available. If you copy the tbmigrate.exe program to another location, make sure to

also copy these libraries. The libraries reside in the installation directory of CCM Designer for Windows, which is typically: `C:\Program Files (x86)\ITPMDK <name>`.

Input files

Text Blocks are loaded from XML files, Rich Text Blocks are loaded from Microsoft Word *.doc or *.docx files. This means that to load text fragments as Text Blocks, you have to prepare them as XML files. To load documents as Rich Text Blocks, no special preparation is necessary, except for collecting them in an input folder for the migration tool.

XML format

For the XSD specifying the structure of the Text Block XMLs, see Examples. If needed, the XSD can also be written to the file TBimport.xsd with the following command line invocation.

```
tbmigrate /xsd
```

A Text Block consists of exactly one `<tbk>` node at top level with the attribute "xsv" to indicate the Text Block version. This `<tbk>` node can contain multiple paragraphs, ordered, and unordered lists of paragraphs, represented by the nodes `<par>` and `<lst>`. The attribute "ordered" on a `<lst>` node indicates whether a list is ordered or unordered. Paragraph nodes can either contain normal text or header text and fields indicated by the attribute "font." A paragraph can also have an indentation, described by the attribute "indentation."

Within paragraphs, you can have multiple texts, Fields, and special characters, stored in the nodes `<txt>`, `<fld>`, and `<chr>`. All nodes have the attributes "bold", "italic", and "underline" to indicate their layout.

The `<chr>` nodes represent a non-breaking space, a non-breaking hyphen, or a line break. This is indicated by the attribute "type." Each `<chr>` can represent only one character.

The attribute "set" in a `<fld>` node indicates the Field Set for the Field. The value of this node is the name of the Field. Each `<fld>` node can represent only one Field.

The `<txt>` nodes just contain text, possibly inside a CDATA section. Note that white space in these nodes is significant and will show up in the resulting Text Block.

The XML should conform to the XSD. Also, Field and Field Set names are limited to characters from the set Latin-1 character and subject to the naming rules of the Template Scripting language.

Preparing files

For each Text Block to be imported, a separate file needs to be created that contains its XML content. The file should have .xml as extension, and its name is used as the name of the Text Block. All Text Block files should be collected in a folder. Subfolders are allowed. Unless the option `/subfolders` is specified, they do not result in a subfolder structure built in the ITP/Model Development Kit.

Before loading, the TBMigrate tool checks the XML files for validity. To check them before attempting a load, use the following command.

```
tbmigrate /check /folder=<folder>
```

<folder> represents the folder containing the Text Block XML files.

The content of Microsoft Word documents in the source folder, which are imported as Rich Text Blocks, is not checked.

Load Text Blocks

Important Using this tool may cause irreversible changes in the CCM Repository installation. Make sure that you make a backup of the database before using it.

```
tbmigrate /load
          /folder=<folder>
          /project=<project>
          [/target=<target folder>]
          [/subfolders]
          [/unlock | /accept]
          [/description=<description>]
          [/server.host=<host> /server.port=<port>]
          [/user=<user> /password=<password> /token=<token>]
```

Use the parameters and flags as follows:

- <folder> Required. Represents the folder containing the Text Block XML files.
- <project> Required. The name of the CCM Repository project into which the Text Blocks are imported. This project must already exist. Text Blocks will be loaded into the top-level folder Text Blocks of this project, while Field Sets are created in the top-level folder Field Sets.
- <target folder> Optional. Specifies the name of a folder beneath the folder Text Blocks. If given, the Text Blocks are loaded into this folder. The folder is created if it does not yet exist. This option has no effect on the location where the Field Sets are stored.
- /subfolders Optional. If present, the program loads Text Blocks into subfolders in CCM Repository according to the subfolders of the input folder where they are loaded from. The folders are created if they do not exist yet. If this flag is omitted, also the Text Blocks found in subfolders of the input folder will be loaded into the folder Text Blocks or to <target folder>. This option has no effect on the location where the Field Sets are stored.
- /unlock and /accept Optional. The flags request that the loaded Text Blocks and Field Sets are unlocked (/unlock) or unlocked and marked as [accepted] (/accept). If neither is used, the loaded objects remain [in development] and locked by the loading user.
- /description=<description> Optional. Specifies a description to be added to the imported objects and revisions. If this flag is not present, the default description "Imported from <filename>" is used. To add no description to the loaded objects, you can leave the parameter <description> empty.
- <host> / <port> Required (usually provided in the itprep.ini file). Provide the host name and port of the CCM Repository server.
- <user> / <password> When the LDAP mode is disabled, specify the user name and password for the CCM Repository user account to perform the import. If you omit the user name and password, the tool attempts to use the current Windows user account. The Text Blocks and Field Sets created or change by the import are locked by the specified or detected user. For more information, see the *Kofax Customer Communications Manager Repository Administrator's Guide*.

- `<token>` When the LDAP mode is enabled, specify the logon token generated with CCM Designer. For more information, see CCM Designer online Help.

The program validates all Text Block XML files before any content is loaded into CCM Repository. If during this check any errors are encountered, no Text Blocks are loaded.

Text Blocks are created if they do not exist yet. If an imported Text Block is already present, a new revision is created for it. In that case, it should not be locked by another user. If the Text Block was locked by the user account used for performing the import, the existing [in development] revision is overwritten.

The Text Blocks are loaded into the top-level folder Text Blocks of the target project, unless `/target` or `/subfolders` is specified. When checking for existing Text Blocks, only those residing in the appropriate target folder are considered.

Text Blocks are searched for Fields and Field Sets.

Fields and Field Sets that do not exist yet are created. Field Sets that are already present receive a new revision. In that case, they should not be locked by another user. If a Field Set was locked by the user account used to perform the import, the existing [in development] revision is overwritten.

The Field Sets are loaded into the top-level folder Field Sets of the target project. When checking for existing Field Sets, only those residing in this top-level folder Field Sets are considered. Usage relationships between Text Blocks, Fields, and Field Sets are created, too.

When the program is finished, all imported objects remain locked, unless the `/unlock` or `/accept` flag is provided.

Rich Text Blocks are not checked for Fields and Field Sets. No Fields and Field Sets are created when loading a Rich Text Block.

The user account must have sufficient authorization to perform all tasks required during the import. The following rights are required:

- Create and edit Text Blocks on the folder Text Blocks
- Create and edit Field Sets on the folder Field Sets
- Edit Text Blocks on pre-existing Text Blocks that are overwritten by the import
- Edit Field Sets on pre-existing Field Sets that are expanded by the import
- Unlock Text Blocks and Field Sets, if the `/unlock` or `/accept` flag is given
- Mark Text Blocks and Field Sets as [accepted], if the `/accept` flag is given
- Add characteristics

After loading, the file migratetextblocks-report.txt is written. This file contains a brief account of the loading process, as well as any errors encountered.

Return codes

If importing or checking is successful, 0 (zero) is returned as error code. The following codes are returned in case of an error.

Error code	Description
1	One or more of the Text Blocks to be imported contain errors.

2	A file name is malformed.
3	A Text Block name is invalid; a file name cannot be used as Text Block name.
4	The validator program tbval.exe was not found. Check that it is in the same folder as tbmigrate.exe.
5	Field Set name is not valid.
6	Field name is not valid.
7	User has insufficient authorization.
8	Unable to unlock object. Usually indicates that someone is using the object.
9	Required command line parameters were missing.
10	An unexpected error occurred. This may include authentication or authorization failure, a missing project, and others.
11	Unable to lock object. Usually indicates that someone is using the object.

Examples

The following is the XSD describing the Text Block XML. It can also be obtained by running tbmigrate.exe with the `/xsd` flag. Then, it will be written in the file TBimport.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.aia-itp.com/Repository/3.1/TextBlockImport"
 elementFormDefault="qualified" xmlns="http://www.aia-itp.com/Repository/3.1/
TextBlockImport" xmlns:mstns="http://www.aia-itp.com/Repository/3.1/TextBlockImport"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="tbk">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="par" type="Paragraph" />
        <xs:element name="lst" type="List" />
      </xs:choice>
      <xs:attribute name="xsv" use="required" fixed="2.0.1" />
    </xs:complexType>
  </xs:element>
  <xs:complexType name="List">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="par" type="Paragraph" />
      <xs:element name="lst" type="List" />
    </xs:choice>
    <xs:attribute name="style" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="unordered" />
          <xs:enumeration value="ordered" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:complexType name="Paragraph">
```

```
   <xs:choice maxOccurs="unbounded">
     <xs:element name="txt" type="Text" minOccurs="0" />
     <xs:element name="fld" type="Field" minOccurs="0" />
     <xs:element name="chr" type="Character" minOccurs="0" fixed="" />
   </xs:choice>
   <xs:attribute name="indentation" use="required" type="xs:nonNegativeInteger" />
   <xs:attribute name="hanging-indentation" use="optional" type="Boolean"
fixed="false" />
   <!-- hanging-indentation is for internal use only -->
   <xs:attribute name="font" use="required">
     <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="normal" />
         <xs:enumeration value="header" />
       </xs:restriction>
     </xs:simpleType>
   </xs:attribute>
 </xs:complexType>
 <xs:complexType name="Text">
   <xs:simpleContent>
     <xs:extension base="xs:string">
       <xs:attributeGroup ref="FontStyle" />
       <xs:attribute name="id" use="optional" type="String254" fixed="" />
       <!-- id is for internal use only -->
     </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
 <xs:complexType name="Field">
   <xs:simpleContent>
     <xs:extension base="String254">
       <xs:attributeGroup ref="FontStyle" />
       <xs:attribute name="set" use="required" type="String254" />
       <xs:attribute name="id" use="optional" type="String254" fixed="" />
       <!-- id is for internal use only -->
     </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
 <xs:complexType name="Character">
   <!-- special/whitespace characters -->
   <xs:simpleContent>
     <xs:extension base="xs:string">
       <xs:attributeGroup ref="FontStyle" />
       <xs:attribute name="type" use="required">
         <xs:simpleType>
           <xs:restriction base="xs:string">
             <xs:enumeration value="LBR" />
             <!-- Line break -->
             <xs:enumeration value="NBSP" />
             <!-- Non-breakable space -->
             <xs:enumeration value="NBHH" />
             <!-- Non-breakable hyphen -->
           </xs:restriction>
         </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="id" use="optional" type="String254" fixed="" />
       <!-- id is for internal use only -->
     </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
 <xs:attributeGroup name="FontStyle">
   <xs:attribute name="underline" use="optional" type="Boolean" default="false" />
   <xs:attribute name="italic" use="optional" type="Boolean" default="false" />
   <xs:attribute name="bold" use="optional" type="Boolean" default="false" />
 </xs:attributeGroup>
```

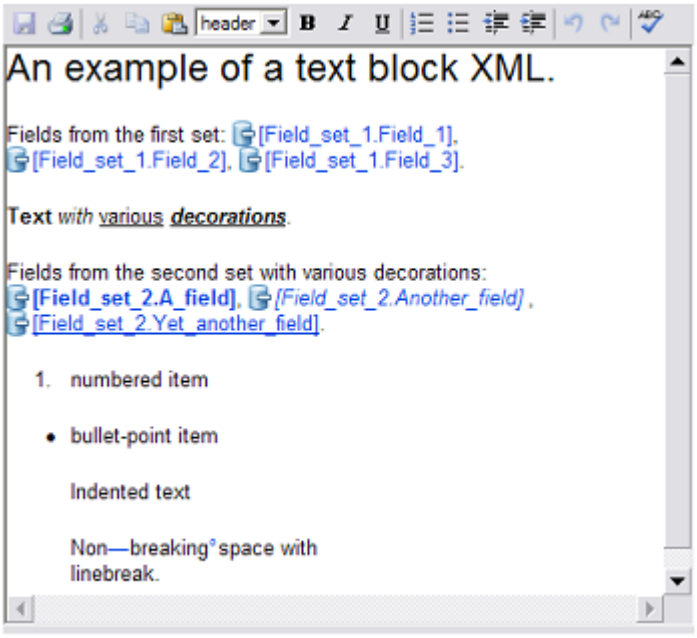
```
  <xs:simpleType name="String254">
    <xs:restriction base="xs:string">
      <xs:maxLength value="254" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="Boolean">
    <xs:restriction base="xs:string">
      <xs:enumeration value="false" />
      <xs:enumeration value="true" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

The following is an example Text Block Import XML.

```
<?xml version="1.0" encoding="utf-8"?>
<tbk xsv="2.0.1" xmlns="http://www.aia-itp.com/Repository/3.1/TextBlockImport">
  <par font="header" indentation="0">
    <txt underline="false" italic="false" bold="false">An example of a text block
 XML.</txt>
  </par>
  <par font="normal" indentation="0">
    <txt underline="false" italic="false" bold="false">Fields from the first set: </
txt>
    <fld underline="false" italic="false" bold="false" set="Field_set_1">Field_1</fld>
    <txt underline="false" italic="false" bold="false"><![CDATA[, ]]></txt>
    <fld underline="false" italic="false" bold="false" set="Field_set_1">Field_2</fld>
    <txt underline="false" italic="false" bold="false"><![CDATA[, ]]></txt>
    <fld underline="false" italic="false" bold="false" set="Field_set_1">Field_3</fld>
    <txt underline="false" italic="false" bold="false"><![CDATA[.]]></txt>
  </par>
  <par font="normal" indentation="0">
   <txt underline="false" italic="false" bold="true">Text</txt>
   <txt underline="false" italic="false" bold="false"><![CDATA[ ]]></txt>
   <txt underline="false" bold="false" italic="true">with</txt>
   <txt underline="false" italic="false" bold="false"><![CDATA[ ]]></txt>
   <txt italic="false" bold="false" underline="true">various</txt>
   <txt underline="false" italic="false" bold="false"><![CDATA[ ]]></txt>
   <txt bold="true" italic="true" underline="true">decorations</txt>
   <txt underline="false" italic="false" bold="false"><![CDATA[.]]></txt>
  </par>
  <par font="normal" indentation="0">
    <txt underline="false" italic="false" bold="false">Fields from the second set with
 various decorations: </txt>
    <fld underline="false" italic="false" bold="true" set="Field_set_2">A_field</fld>
    <txt underline="false" italic="false" bold="false"><![CDATA[, ]]></txt>
    <fld underline="false" bold="false" italic="true" set="Field_set_2">Another_field</
fld>
    <txt underline="false" italic="false" bold="false"><![CDATA[, ]]></txt>
    <fld italic="false" bold="false" underline="true"
 set="Field_set_2">Yet_another_field</fld>
    <txt underline="false" italic="false" bold="false"><![CDATA[.]]></txt>
  </par>
  <lst style="ordered">
    <par font="normal" indentation="0">
      <txt underline="false" italic="false" bold="false">numbered item</txt>
    </par>
  </lst>
  <lst style="unordered">
    <par font="normal" indentation="0">
      <txt underline="false" italic="false" bold="false">bullet-point item</txt>
    </par>
  </lst>
  <par font="normal" indentation="1">
```

```
    <txt underline="false" italic="false" bold="false">Indented text</txt>
  </par>
  <par font="normal" indentation="1">
    <txt underline="false" italic="false" bold="false">Non</txt>
    <chr underline="false" italic="false" bold="false" type="NBHH"/>
    <txt underline="false" italic="false" bold="false">breaking</txt>
    <chr underline="false" italic="false" bold="false" type="NBSP"/>
    <txt underline="false" italic="false" bold="false">space witth</txt>
    <chr underline="false" italic="false" bold="false" type="LBR"/>
    <txt underline="false" italic="false" bold="false">linebreak.</txt>
  </par>
</tbk>
```

The preceding example, being imported into CCM Repository, results in the following Text Block.



Chapter 3

Import Quick Templates

To import Quick Templates in CCM Repository, you can use the ImportQuickTemplates tool, which resides in `<deploy root>\CCM\Programs\<version>\ITPMDKRepositoryServer`. After import, they become visible in CCM Designer.

```
ImportQuickTemplates /file=<importxml> /user=<user> /password=<password> /token=<token>
                                              [/verbose] [/logfile=<logfile>]
```

The command has these parameters:

- `file` Required. Indicates an XML file containing the Quick Templates to import as well as a number of other settings.
- `user` / `password`. Only required when the LDAP mode is disabled. A valid CCM Repository user and password.
- `token` Only required when the LDAP mode is enabled. Logon token generated with CCM Designer. For more information, see CCM Designer online Help.
- `verbose` Optional. Shows more information on the ImportQuickTemplates tool while running the application.
- `logfile` Optional. Name of the log file. If not given, it is importquicktemplates.log by default.

The following is an example of the XML structure.

```
<ImportQuicktemplates
  xmlns = " http://www.kofax.com/ccm/importQuicktemplates/1.0"
  version =  required: {version}
  project  =   required: {project}
  uri   =   required: {base URL to use for the restapi calls}
  promote  =   optional: {status to promote to}
  createDocumentTemplate = optional: {"true" or "false"}.
  If "true" document templates will be created for all imported quick templates.
  If not present or "false" the document templates will not be created
  exist  =  optional: {"fail", "newrevision", "overwrite"}
  continueOnError      = optional: {"true" or "false"}
  If "true" Log the failure and continue importing}>
<Quicktemplate name={Quick Template name}>{document location}</Quicktemplate>
..
<Quicktemplate name={Quick Template name}>{document location}</Quicktemplate>
</ImportQuicktemplates>
```

The attributes have the following meaning:

- `version={version}` Indicates the version of the import XML. It can be used to differentiate between various versions of the import XML. Must be set to 1.
- `project={project}` Name of the project where the Quick Templates are loaded.
- `uri={base url to use for the restapi calls}`This is the URL to use to connect to CCM Repository for your instance.

To retrieve the correct value for `uri=`, open the itpmdkapi.ini file that resides in the Config folder of the instance: `[drive:]\CCM\Work\<CCM_version>\instance_<#>\designer\Config`.

Look for the line `address=` and copy the value. In this example, the value is `http://localhost/cma_designer_01_5.2`.

```
[Configuration]
logfile=C:\CCM\Work\5.2\Instance_01\designer\Log\TESTCCM50S3\CMA
 [designer_01_5.2]\cmapi.log
address=http://localhost/cma_designer_01_5.2
MaxItemsInObjectGraph=500000
```

- `promote={status to promote to}` Specifies the status to be assigned to the imported Quick Templates, either draft, accepted, or published. If this attribute is omitted, the imported Quick Templates are not imported. Depending on `createDocumentTemplate`, this also applies to the created Document Templates.
- `createDocumentTemplate={"true" or "false"}` Default value is `"false"`. If the value is `"true"`, Document Templates will be generated for the imported Quick Templates if they did not exist before. If the Quick Template already existed, no Document Template is generated.
- `exist={"fail", "newrevision", "overwrite"}` Default value is `"fail"`. Indicates what action to take when the Quick Template to be imported already exists.

  `"fail"` indicates that the import fails when the Quick Template already exists. `"newrevision"` indicates that a new revision will be created. Quick Templates that where in development are unlocked prior to creating a new revision. Quick Templates that where locked by a user other than the importer fail. `"overwrite"` indicates a behavior similar to that of the new revision, except for the Quick Template that was in development and locked by the importer. In that case, the content of the Quick Template is overwritten.
- `continueOnError={"true" or "false"}` Indicates what to do when a failure is encountered. If set to `"true"`, the import continues but all failed imports are logged. If set to `"false"`, the import stops at the first failure that is encountered.