Kofax Customer Communications Manager

5.0

Developer's Guide

CCM Repository 5.0

KOFAX
from Lexmark

Contents

Introduction to the CCM Repository Developers Guide

The Developer's Guide describes the use of API's and automation in the ITP/MDK Microsoft Word environment.

The chapter List Objects API (page 4) discusses how to set-up and install the List Objects API. The third chapter, Get Model API (page 9), discusses how an API is able to retrieve an ITP Model from the CCM Repository without a deploy.

The fourth chapter, Text Block migration (page 12), explains the tool TBMigrate, which is created in order to load text fragments from third-party applications into the CCM Repository as (Rich) Text Blocks. The last chapter, Automation in the ITP/MDK Microsoft Word environment (page 17), explains how you can allow Microsoft Word templates to extract information from documents opened from within theCCM Repository.

The Appendix (page 19) contains the XSD describing the Text Block XML.

All documentation, with the exception of the Installation Guide and the Release Notes, can be found on the ITP Knowledge Center. The Installation Guide and Release Notes are only available for customers who have access to the support section of the corporate website.

Copyrights and trademarks

List Objects API

The API List Objects lets you extract a list of objects contained in a project in an CCM Repository installation. Optionally, you can specify a path in that project; the result will then be limited to objects residing below that path. This API only lists ITP Models, Text Blocks, and Content Wizards. You can choose which revisions are listed by specifying [development], [current], [accepted], or [published] as value of the optional level parameter.

When this parameter is not specified the listed revision depends on the value of the compatibility option "Handle accepted revisions as published". Refer to the CCM Repository Administrative Manual chapter Administrative tasks for more information on handling [accepted] revisions as [published] and chapter Administrator menus for more information on Installation properties.

Set-up

The API consists of the client program listobjects.exe that will connect to the CCM Repository Server for collecting the requested data. The computer running this API program therefore needs network access to the one running the CCM Repository Server. It also needs to be of the same version of the CCM Repository Server. If this server is upgraded, you must also use the new version of the List Objects API.

The program listobjects.exe can be found in the subdirectory Extra of the installation directory of the CCM Repository Server. Copy the program to the location where you want to use it. Optionally, you can create a file listobjects.ini accompanying it.

The listobjects.exe program additionally needs the libraries cc32160mt.dll, borlndmm.dll, aiacore.dll, and itprepbase.dll to be available. If you copy the listobjects.exe program to another location, make sure to also copy these libraries. The libraries can be found in the CCM Repository server installation directory.

Invocation

The syntax for calling the List Objects API client program is as follows:

```
listobjects /project=<project name>
            /file=<output xml file>
            [/path=<path in project>]
            [/server.host=<server host name>]
            [/server.port=<server port>]
            [/cfg=<configuration file>]
            [/level=<required revision level>]
            [/user=<required user for development revision>]
```

The parameters can be given in any order and have the following meaning:

- **project**; this mandatory parameter specifies the name of the project in the ITP/MDK Repository for which the objects must be listed.
- **file**; this mandatory parameter describes the name of the output file. The XML listing of the objects is written in this file.
- **path**; this parameter is optional. If given, it restricts the objects listed to those that reside under the given folder path in the project. To restrict the output to Text Blocks, specify "Text Blocks" as path; to restrict the output to a specific Text Block folder, make sure the path starts with "Text Blocks". Likewise, output can be restricted to Content Wizards by specifying a path beginning with "Content Wizards".
- **server.host** and **server.port** point out which server to connect to. These parameters are mandatory, if not specified either on the command line or in the configuration file, the API will fail.
- **cfg**; this parameter optionally specifies a configuration file to read. If omitted, the program will try to read listobjects.ini, if it exists. To be found, the file listobjects.ini should reside in the same directory as the listobjects.exe program, or its path should be included in the name.
- **level**; this parameter specifies a minimal level of revisions, each object that has got such a revision or a revision in a higher level will be listed. The parameter is optional, if not specified the compatibility option is set to "Handle accepted revisions as published", just the accepted revisions will be listed, otherwise just the published revisions.
- **user**; this parameter is only necessary, if you want to list the development revision. For this the user is needed that locked the required objects.

You can also specify these parameters, all or in part, in a configuration file, by default in the file listobjects.ini. If the parameter cfg is not present on the command line, the program will automatically read the file listobjects.ini if it exists in the same directory as the listobjects.exe. Parameters on the command line will always have priority over parameters specified in a configuration file. This file should look like:

```
[Configuration]
project=<project name>
path=<path in project>
file=<output xml file>
level=<required revision level>
user=<required user for development revision>
[Server]
host=<server host name>
port=<server port>
```

Invocation examples:

```
listobjects /project="my project" /file=output.xml
listobjects /project="another project" /path="text blocks"
/file=textblocks.xml
listobjects /project="my project" /path="model docs\invoices"
/file=models.xml
listobjects /project="my project" /file=output.xml /cfg=itprep.ini
listobjects /project="my project" /file=output.xml /server.host=server01
listobjects /project="my project" /file=output.xml /level=accepted
listobjects /project="my project" /file=output.xml /level=development
/user=jane
```

Return codes

The List Objects API client program returns the following codes to indicate problems:

Code	Problem
1	The command line parameters are incomplete.
2	The given path is not found within the given project.
3	The given project is not found within the CCM Repository.
4	The result XML file cannot be written.
5	Something else went wrong.
6	The required level is not valid
7	The required level is [development] and there is no user provided
8	The user is not valid

When an error occurs, an error description is written to standard error.

XML specification

The generated XML contains a version element and a project element. The XML can be validated against the List Objects XSD. The file listobjects.xsd is located in the directory of the List Objects API client program, which can be found in the subdirectory Extra of the installation directory of the CCM Repository Server.

XML structure

The element **listobjects** contains the **version** (as an attribute) of the API List Objects used to generate this XML, and a single **project**.

The element **version** is a decimal number that specifies the version of the API that generated the XML. This version currently is 1.1, the older 1.0 version did not contain Content Wizards.

The element **project** contains exactly one occurrence of a **name**, a **description**, and possibly the elements **textblocks** and **contentwizards**. It also can contain **folder** and **model** elements. ITP Models, Text Blocks, and Content Wizards can respectively be found (possibly nested) only in folders, Text Block folders, and Content Wizards folders. Like the CCM Repository, they cannot occur mixed. Furthermore, Text Blocks and Text Block folders are contained within the element **textblocks**, and Content Wizards and Content Wizard folders are contained in the **contentwizards** element.

The element **folder** contains exactly one occurrence of a **name** and a **description**. It can contain any number of **model** and **folder** elements, but no Text Blocks or Text Block folders.

The element **model** contains exactly one occurrence of a **name** and a **description**. The existence of such an element means that there is a published revision of the ITP Model that can be run by this name. Refer to the topic List objects API (page 4) for more information on the accepted and published mark in revisions.

The element **textblocks** occurs exactly once and contains a number of **textblock** and **textblockfolder** elements.

The element **textblockfolder** contains exactly one occurrence of a **name** and a **description**. It can contain any number of **textblock** and **textblockfolder** elements.

The element **textblock** contains exactly one occurrence of a **name** and a **description**. The existence of such an element means that there is a published revision of the Text Block with this name that can be used by the ITP Models of this project. The element **contentwizards** occurs exactly once and contains a number of **contentwizard** and **contentwizardfolder** elements.

The element **contentwizardfolder** contains exactly one occurrence of a **name** and a **description**. It can contain any number of **contentwizard** and **contentwizardfolder** elements.

The element **contentwizard** contains exactly one occurrence of a **name** and a **description**. The existence of such an element means that there is a published revision of the Content Wizard with this name that can be used by the ITP Models of this project.

The element **description** is not optional but can be empty and has at most 1999 characters.

The element **name** is never empty and has at most 254 characters.

XML example

This is the generated result for an example project named Northwind List Objects API Example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<listobjects version="1.1"
xmlns="http://www.aia-itp.com/Repository/3.1/ListObjects"
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance">
  <project>
    <name>Northwind List Objects API Example</name>
    <description>Demonstration project</description>
    <folder>
      <name>Model Documents</name>
      <description />
      <folder>
        <name>Include</name>
        <description />
        <folder>
          <name>forms</name>
          <description />
        </folder>
      </folder>
      <folder>
        <name>Mailing</name>
        <description />
        <model>
          <name>mailing_to_all_customers</name>
          <description />
        </model>
```

```
        <model>
          <name>mailing_to_a_customer</name>
          <description />
        </model>
      </folder>
    </folder>
    <textblocks>
      <textblockfolder>
        <name>Text Blocks for Mailing</name>
        <description />
        <textblock>
          <name>SpecialOffer</name>
          <description>Special offer Q1 2006</description>
        </textblock>
        <textblock>
          <name>VerifyData</name>
          <description>Verify company data</description>
        </textblock>
      </textblockfolder>
    </textblocks>
    <contentwizards>
      <contentwizard>
        <name>Mailing</name>
        <description />
      </contentwizard>
    </contentwizards>
  </project>
</listobjects>
```

This is a selection of the previous example using the path *Text blocks\Text blocks for Mailing*:

```
<?xml version="1.0" encoding="UTF-8" ?>
<listobjects version="1.1"
xmlns="http://www.aia-itp.com/Repository/3.1/ListObjects"
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance">
  <project>
    <name>Northwind List Objects API Example</name>
    <description>Demonstration project</description>
    <textblocks>
      <textblockfolder>
        <name>Text Blocks for Mailing</name>
        <description />
        <textblock>
          <name>SpecialOffer</name>
          <description>Special offer Q1 2006</description>
        </textblock>
        <textblock>
          <name>VerifyData</name>
          <description>Verify company data</description>
        </textblock>
      </textblockfolder>
    </textblocks>
  </project>
</listobjects>
```

Get Model API

An API is available to retrieve a model from the CCM Repository without a deploy. It will store the model in a folder specified by the requesting client.

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

The above code is one line.

The required parameters are:

<model name> is the name (path) of the model to be retrieved. It is the full path of its model document in the CCM Repository i.e., project\folder\model document.

Example

```
-model=\MyProject\MyFolder\MyModeldocument
or
-model="\MyProject\MyFolder itp\some model document"
```

The quotes are necessary because of the spaces in the model's name and path.

<model file> is the name of the file in which the model will be stored. Use double quotes if there is a space in the file's name or path.
The model file name must be specified with an absolute path. Only when deploying for ITP/Workstation, the model file name may be a relative path. In that case it is resolved with respect to the Models folder as specified in the active file ITP configuration of ITP/Workstation.

<repository ini file> is the path to the repository ini file.
The ini file is needed for the connection to the CCM Repository Server. It contains the (mandatory) server host and port.

Choose one of the options:

```
-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 [current] revision is retrieved. If the model doesn't have a [current] revision the CCM Repository will respond with an error.

Instead of a revision number, also the text "accepted" or "published" may be passed to the -r option (e.g., -r=accepted). This will retrieve the [accepted] or [published] model revision respectively, if present.

Documents and folders that are marked for deletion are disregarded when looking for the model.

Set-up

The GetModel.exe program additionally requires the libraries cc32160mt.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.

Return codes

The following error codes can be returned by the GetModel API:

5	The specified model does not exist in the CCM Repository
6	There is no revision with the given revision number for the specified model
7	The specified label does not exist in the CCM Repository
8	There is no revision with the given label for the specified model
9	The model specified does not have a [current] revision
10	The model specified does not have an [accepted] revision
11	Duplicate revisions have been found for the specified model and revision
12	The model 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 Model file
23	Failed to write Model 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

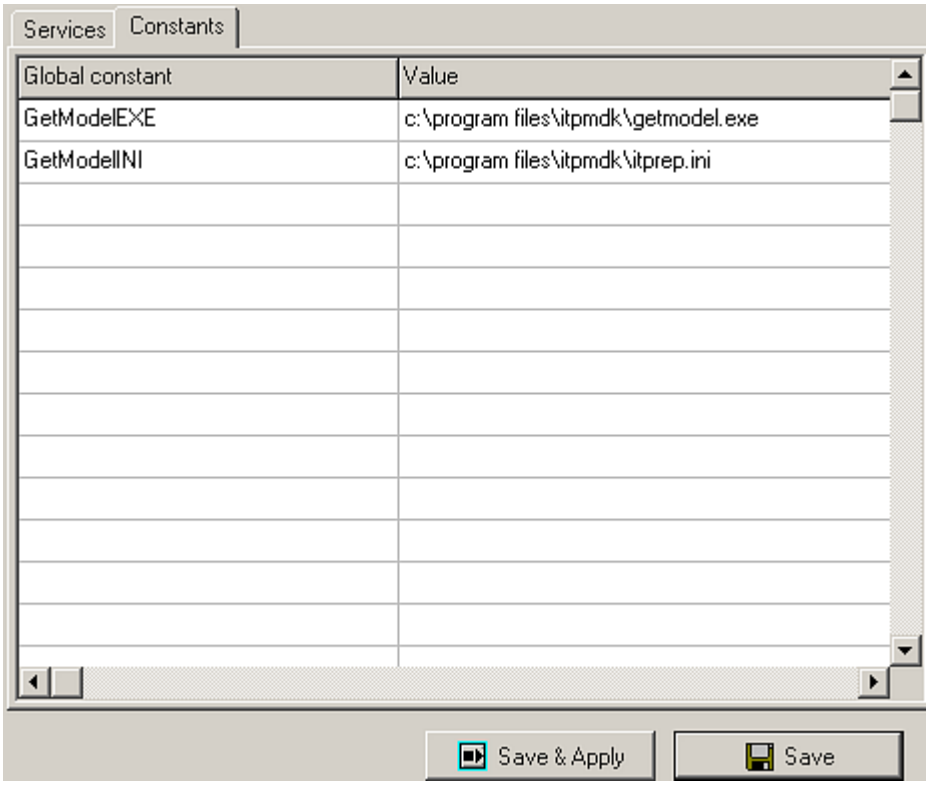
CCM Core

An example script for CCM Core, showing how to call GetModel.exe from a script, is available. It can be found in the installation folder of the CCM Repository Client. Copy this script to the scripts folder of an CCM Core installation. Start the CCM Core Administrator. On the Services tab, click **`Add service...`** and enter GetModel as service name. The script editor opens with the script. You can close the editor if you do not want to make changes to it.

The script needs two constants to set on the Constants tab:

1. GetModelEXE; the path, including the file name, to the GetModel.exe API program.
2. GetModelINI; the path, including the file name, to a CCM Repository Client configuration file (specifies the server to connect to).

Example



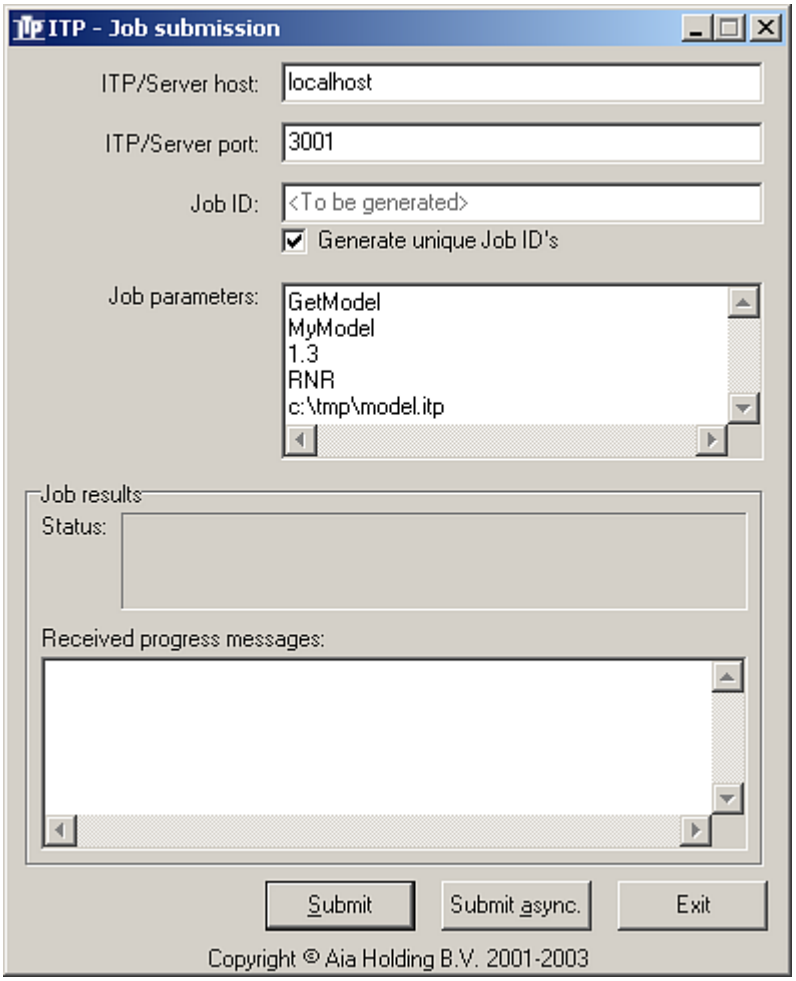
Save and compile, and then click **Save & Apply** to restart the affected CCM Core components.

Using the CCM Core service

The script takes four parameters:

1. the model name
2. a revision identification; this is either a revision number or a label, depending on the next parameter
3. identification type, which is RNR or LBL; RNR specifies that the preceding revision identification is a revision number, LBL indicates that it is a label
4. the model file name under which the retrieved model is sent to the client

An example with the test tool (which takes the service name as the first entry in the Job parameters box):



Text Block migration

The tool TBMigrate has been created in order to load text fragments from third-party applications into the CCM Repository as Text Blocks or Rich Text Blocks. TBMigrate only manages (Rich) Text Blocks, not Forms or Views. Text Blocks must be present in the XML format used by the 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 the 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, e.g. in the Content Wizard Editor or in an CCM ComposerUI Server model run, the text "Preview not available" is shown. To generate a proper preview, edit the imported Rich Text Block and save it.

On the other hand, Rich Text Blocks have the optional extra functionality that you can 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. Currently supported is the addition of characteristics to a Rich Text Block. This is done by including a line "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 on the fly when Fields are referenced in the Text Blocks. This behavior can be used if it is needed 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 TBMigrate the dummy Text Block can be deleted from the CCM Repository.

Set-up

The tool Text Block Migration consists of the program tbmigrate.exe and a helper program tbval.exe. They can be found in the subdirectory "extra" of the CCM Repository Client installation directory. 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 Repository Client installation directory. Alternatively, copy or create an itprep.ini file in the directory "extra".

The tbmigrate.exe program additionally needs the libraries cc32160mt.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. You will also need to do so if you want to run the program in the "extra" directory. The libraries can be found in the CCM Repository client installation directory.

Input files

Text Blocks are loaded from XML files, Rich Text Blocks are loaded from Microsoft Word *.doc or *.docx files. This means that if you want to load text fragments as (classic) Text Blocks, you have to prepare them as XML files. This preparation is explained below. 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

The XSD specifying the structure of the Text Block XMLs is given in the Appendix. Also, an example can be found there. If needed, the XSD can also be written to the file TBimport.xsd with the following command line invocation:

```
tbmigrate /xsd
```

An ITP 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 and (un)ordered 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 from which Field Set a Field comes. The value of this node is the name of the Field. Each <fld> node can represent only one Field.

The <txt> nodes just contain their 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 ITP 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 will check the XML files for validity. If you want to check them before attempting a load, use the following command:

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

Here, <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.

Loading Text Blocks

> **Warning**
> Using this tool may cause many irreversible changes in the CCM Repository installation. Make sure you have made a backup of the database before using it.

The TBMigrate tool for Text Block migration is a client-side program. You should run it from within an CCM Repository Client folder. The itprep.ini file stored there is needed to be able to connect to the server. Use the following command:

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

In this command:

- <folder> represents the folder containing the Text Block XML files.
- <project> is 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, and Field Sets are created in the top-level folder Field Sets.
- <target folder> optionally specifies the name of a folder beneath the folder Text Blocks. If given, the Text Blocks are loaded into this folder, instead of into the folder Text Blocks itself. 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 is an optional parameter. If present, the program will load Text Blocks into subfolders in the 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 the <target folder>. This option has no effect on the location where the Field Sets are stored.
- /unlock and /accept are optional flags. They request that the loaded Text Blocks and Field Sets are unlocked (/unlock) or unlocked and marked as [accepted] (/accept). If neither is given, the loaded objects remain [in development] and locked by the loading user.
- With the optional parameter /description=<description> you can specify a description that is 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> and <port> identify the CCM Repository server to connect with. These parameters are mandatory, but normally provided in the itprep.ini file.
- <user> and <password> identify the CCM Repository user account that is used to perform the import. Text Blocks and Field Sets created or changed by the import are locked by this user. <user> and <password> may be omitted. In that case the tool tries to perform a unified logon based on the Windows user ID. See the Administrative Manual, chapter User management for details.

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

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

import, the existing [in development] revision will be 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 finishes, 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 will be 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.
- The right to unlock Text Blocks and Field Sets, if the /unlock or /accept flag is given.
- The right to mark Text Blocks and Field Sets as [accepted], if the /accept flag is given.
- The right to 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:

1 - One or more of the Text Blocks to be imported contain(s) 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, e.g., by editing it.

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, e.g., by editing it.

Automation in the ITP/MDK Microsoft Word environment

The CCM Repository has features that allow Microsoft Word templates to extract information from documents opened from within the CCM Repository. These features are implemented in the ITPMDKServer.dot.

> **Note**
> This does not apply to OpenOffice.org documents.

Information extraction

The information that can be retrieved is;

- whether the document is opened from the CCM Repository,
- the display name of the document when the document is opened from the CCM Repository,
- the path to the document in the CCM Repository,
- the read-only state of the document.

4 macros are available in ITPMDKServer.dot for this purpose:

```
Public Sub IsITPMDKDocument(pDocument As Document, pResult As Boolean)
Public Sub GetName(pDocument As Document, pResult As String)
Public Sub GetPath(pDocument As Document, pResult As String)
Public Sub IsReadOnly(pDocument As Document, pResult As Boolean)
```

Each of these macros accepts a Document object as an input parameter and returns its result through the pResult output parameter.

Detect changes

On a number of occasions the ITP/Model Development Kit will make an attempt to call a macro called ITPMDKDocumentChange. This is done whenever:

- The application object fires a DocumentChange event.
- The ITP/Model Development Kit has finished the process of opening a document.
- The information that is stored with document changes.

This macro can be implemented by an external template. It can be called multiple times for one document.

Example.dot

The CCM Repository is shipped with an example template that takes advantage of the features described above. This template is called Example.dot. It can be found in the root of the installation of the CCM Repository Client. This template can be used both as an example and as a framework on which the required functionality can be based.

> **Note**
> Because this functionality should be globally active in the IITP/Model Development Kit Microsoft Word environment, the template should be stored in a StartUp folder of Microsoft Word.

Modules

The Example.dot template contains 2 modules; Custom and ITPMDK. In principle only the Custom module needs to be adapted by you. The ITPMDK module implements the ITPMDKDocumentChange macro. Whenever this macro is called, it retrieves document information on the active document and passes this information by calling external macros that are defined in the Custom module.

Macros

The macros present in the Custom module are:

```
Public Sub GeneralDocument(pDoc As Document)
Public Sub ITPMDKDocument(pDoc As Document, pDocName As String, pDocPath As
String, pDocReadOnly As Boolean)
Public Sub HandleError(pError As ErrObject)
```

GeneralDocument is called by ITPMDKDocumentChange if the active document cannot be identified as being opened from the CCM Repository. ITPMDKDocument is called whenever the active document is identified as being opened from the CCM Repository - the document information is passed through the parameters. HandleError is called whenever an error occurs in the ITPMDKDocumentChange macro - the error object is passed as a parameter.

Public function

Along with this, the ITPMDK module defines a public function through which document information can be retrieved at any time:

```
Public Sub GetDocumentInfo(pDocument As Document,
                           pIsITPMDKDoc As Boolean,
                           pDocName As String,
                           pDocPath As String,
                           pIsReadOnly As Boolean)
```

Implementing the functionality

Given the Example.dot template, the following steps are enough to implement additional functionality in the ITP/Model Development Kit Microsoft Word environment:

1. Create a copy of the Example.dot template.
2. Open this copy.
3. Customize the template copy (e.g., add menus, toolbars etcetera.).
4. Implement the macros in the Custom module from the Visual Basic Editor (e.g., enable/disable menu items, buttons etcetera.).

> **Note**
> Because activation of this macros is the consequence of OLE calls from the CCM Repository, it is very important that their implementation should never be blocking, i.e., no message boxes or modal forms should be shown as a result of them.

5. Move the macro to a StartUp folder of your Microsoft Word installation.

When, after this, a document is viewed or edited from the CCM Repository, the template should be available and its functionality should be active.

Appendix

XSD

This is the XSD describing the Text Block XML. It can also be obtained by running tbmigrate.exe with the /xsd flag. It will then 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>
```

Example

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>
```

Imported into the CCM Repository this will result in the following Text Block.