Kofax Communications Manager

Getting Started Guide

Version: 5.4.0

Date: 2020-08-26

KOFAX

Table of Contents

Preface

This guide explains how to use Contract Manager to manage instances of Kofax Communications Manager.

Related documentation

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

https://docshield.kofax.com/Portal/Products/KCM/5.4.0-cli2a1c07m/KCM.htm

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

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

Kofax Communications Manager Technical Specifications
Provides information on supported operating system and other system requirement for Kofax Communications Manager.

Kofax Communications Manager Installation Guide
Contains instructions on installing and configuring Kofax Communications Manager and its components.

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

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

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

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

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

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

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

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

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

Kofax 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 Communications Manager ComposerUI for ASP.NET and J2EE Customization Guide
Describes the customization options for KCM ComposerUI for ASP.NET and J2EE.

Kofax Communications Manager ComposerUI for ASP.NET Developer's Guide
Describes the structure and configuration of KCM ComposerUI for ASP.NET.

Kofax Communications Manager ComposerUI for J2EE Developer's Guide
Describes JSP pages and lists custom tugs defined by KCM ComposerUI for J2EE.

Kofax 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 Communications Manager.

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

Getting help with Kofax products

The Kofax Knowledge Base repository contains articles that are updated on a regular basis to keep you informed about Kofax products. We encourage you to use the Knowledge Base to obtain answers to your product questions.

To access the Kofax Knowledge Base, go to the Kofax website and select **Support** on the home page.

Note The Kofax Knowledge Base is optimized for use with Google Chrome, Mozilla Firefox or Microsoft Edge.

The Kofax Knowledge Base provides:

- Powerful search capabilities to help you quickly locate the information you need.

  Type your search terms or phrase into the **Search** box, and then click the search icon.
- Product information, configuration details and documentation, including release news.

  Scroll through the Kofax Knowledge Base home page to locate a product family. Then click a product family name to view a list of related articles. Please note that some product families require a valid Kofax Portal login to view related articles.
- Access to the Kofax Customer Portal (for eligible customers).

  Click the **Customer Support** link at the top of the page, and then click **Log in to the Customer Portal**.
- Access to the Kofax Partner Portal (for eligible partners).

  Click the **Partner Support** link at the top of the page, and then click **Log in to the Partner Portal**.
- Access to Kofax support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.

  Scroll to the **General Support** section, click **Support Details**, and then select the appropriate tab.

Chapter 1

Introduction

Kofax Communications Manager, also known as KCM, is a single integrated solution to automatically produce large numbers of standard documents and compose professional individual correspondence.

KCM is a solution that empowers the organization to engage and communicate with customers by generating relevant and personal communication delivered through different channels.

KCM installation package and on-premise installation

By default, the Kofax Communications Manager installation package deploys a single instance of the KCM software consisting of KCM Core, KCM Repository, KCM Designer, KCM ComposerUI for HTML5, and the Contract Manager. The installation creates one instance ready for use, but you can append additional KCM instances if required.

To manage connections to the instances, you can use the Contract Manager. The Contract Manager is configured with the KCM Local contract that will be associated with instance 1. You might not need to change the Contract Manager configuration.

By leveraging the installation package predefined setup options, you can deploy the package without the Contract Manager or default contracts, or you can install multiple instances of the software on multiple computers.

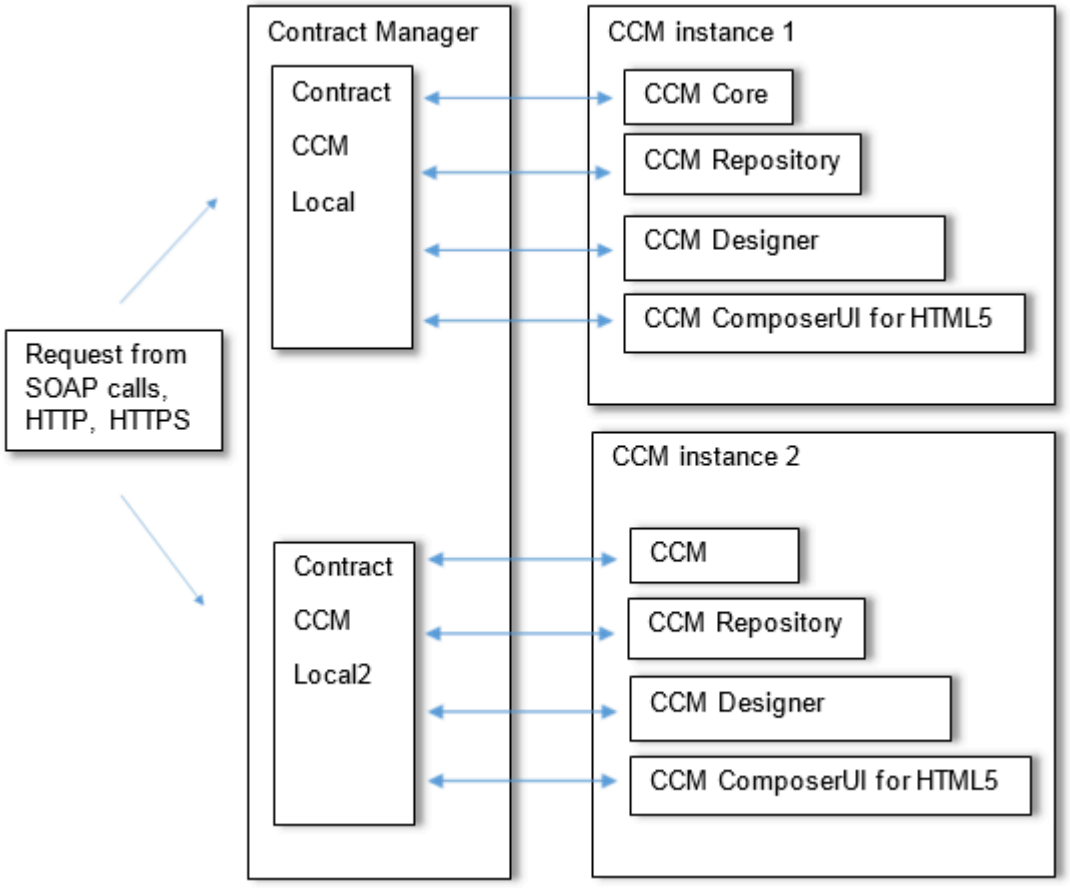
Chapter 2

Work with the Contract Manager

This chapter describes how you can use the Contract Manager and provides links to important step-by-step procedures.

The Contract Manager is designed to manage connections to KCM instances. A KCM instance consists of a KCM Core instance, a KCM Repository instance, a KCM Designer instance, and an instance of KCM ComposerUI for HTML5.

This chart illustrates how the work of the Contract Manager is organized.



The Contract Manager collects and stores information about instances. An **instance** is a collection of KCM product instances that work together to form a KCM instance. To control instance information in the Contract Manager with the ManageCM tool, use the following commands:

- RegisterInstance
- ListInstance
- ChangeInstance
- RemoveInstance

To connect to an instance, you need a contract. A **contract** is defined by a partner and a customer and comes with a number of contract types.

You can use a partner to indicate a partner of the organization, such as a reseller, or to group a number of customers under a descriptive name. A **default partner** called KCM is created during the default installation. You cannot remove the default partner, but you can add extra partners if needed. To use the ManageCM tool to control the partners, use the following commands:

- CreatePartner
- RemovePartner
- ListPartner

A **default customer** called Local is created during the installation. The KCM partner and the Local customer together indicate the KCM Local contract. To create or remove a customer, use the following ManageCM tool commands:

- CreateContract
- ListContract
- ChangeContract
- RemoveContract

The **default contracts** from the standard installation generate the following customers in line with the instance chosen number: Local, Local2, Local3, and so on.

To learn how to replace existing standard contracts, add new contracts to the Contract Manager, and specify partners and customers, see Replace an existing contract by a new contract and Add a new contract to the Contract Manager.

A KCM installation contains a number of **default contract types**. A contract type is a set of interfaces that you can use to access the KCM products. A KCM instance with a contract must support the interfaces in all the available contract types. To use the ManageCM tool to add and remove contract types, use the following commands:

- AddContractTypeToContract
- RemoveContractTypeFromContract

To add a contract type to the Contract Manager and update existing contract types, see Add a new contract type to the Contract Manager and Update an existing contract type, respectively.

Also, you can add **your own contract types and interfaces**, if needed. When you add a new interface, it must be able to access the Core script configured for it. This Core script must be supported for the instance associated with the contract that contains this interface. To add and remove contract types and interfaces, use the following ManageCM tool commands:

- AddInterface

- AddContractType
- RemoveInterface
- RemoveContractType
- ListInterface
- ListContractType

To learn how to add your own contract types and create your own interfaces, see Update an existing contract type, Add a new interface to the Contract Manager, and Create and deploy interface implementation, respectively.

To access KCM products with a contract, you need to **associate an instance with a contract**. You can have more than one instance associated with a contract, but only one can be active. Previous instances are kept to facilitate a possible rollback. Also, you can remove all instance associations from a contract. You can associate instances, activate specific instances, or remove an association using the following commands:

- AssociateInstance
- ActivateInstance
- RollBackInstance
- RemoveInstanceAssociation

You can export configuration entries stored in the Contract Manager to import this configuration again in another Contract Manager. You cannot manage the exported configuration entries prior to importing them. Not all changes in the configuration made using ManageCM are automatically active. To notify the Contract Manager to reload its configuration, use the following commands:

- ReloadConfiguration

The Contract Manager is controlled using the ManageCM tool. For more information, see Use the ManageCM tool.

Enable LDAP mode for KCM Designer

To configure KCM Designer to authenticate users on a configured LDAP server, follow these steps:

1. Create an LDAP properties file consisting of key=value pairs.
2. Copy the LDAP properties file to:
   `<deploy root>\KCM\Work\<version>\ContractManager\Config\ldap.properties`
   You cannot change this location.
3. Restart the Contract Manager.
   Once the Contract Manager is restarted, its log file indicates whether the configuration file can be loaded and a list of available groups can be retrieved.
4. Start KCM Designer.
   Authentication now takes place on the configured LDAP server.

LDAP properties file

Define the following key=value pairs in the LDAP properties file:

- `Connection.Host=` *Required*. Host name or IP address of the LDAP server.
- `Connection.Port=`*Required*. Port number that the LDAP server listens to.
- `Connection.Security=` *Optional*. Security protocol used to encrypt the traffic to the LDAP server. Possible values: `TLS` and `LDAPS`. If no value is specified, the traffic is not encrypted.
- `Prebind.Anonymous=` *Required*. Indicates if initial (before a user logs in) LDAP queries use anonymous bind or a configured LDAP administrative account. Possible values: `True` and `False`.
- `Prebind.User=` Only required when `Prebind.Anonymous` is set to `False`. Distinguished name of the administrative LDAP account used to bind initial LDAP queries.
- `Prebind.Password=` Only required when `Prebind.Anonymous` is set to `False`. Password of the administrative LDAP account used to bind initial LDAP queries.
- `Groups.Base=` *Required*. Distinguished name of the LDAP base entry where all LDAP group entries must be located. These group entries are then used for authorization in KCM Designer. Unless filtered with `Groups.Filter`, the base entry itself is also available as a group for authorization.
- `Groups.Filter=` *Required*. LDAP filter expression that determines which LDAP entries under `Groups.Base` are available as groups for authorization in KCM Designer. Entries that do not conform to the filter are ignored.
- `Groups.NameAttribute=` *Required*. Attribute for a group entry that denotes its name.
- `Groups.MemberAttribute=` *Optional*. Attribute of a group entry that enumerates user members of the group. If no value is specified, group membership is determined by `Users.MemberAttribute`.

  Note `Groups.MemberAttribute` does not support nested groups.

- `Groups.Admin=` *Required*. Distinguished name of the LDAP group entry where all user members are assigned administrative access.
- `Users.Base=` *Required*. Distinguished name of the LDAP entry where all LDAP user entries must be located. These user entries are then used for logging in to KCM Designer. Unless filtered with `Users.Filter`, the base entry itself is also available as a user for logging in to KCM Designer.
- `Users.Filter=` *Required*. LDAP filter expression that determines which LDAP entries under `Users.Base` are available for logging in to KCM Designer. Entries that do not conform to the filter are ignored.
- `Users.NameAttribute=` *Required*. Attribute of a user entry that denotes its name.
- `Users.MemberAttribute=` *Optional*. Attribute of a user entry that enumerates the groups that a user is a member of. If no value is specified, group membership is determined by `Groups.MemberAttribute`. `Users.MemberAttribute` relies on the groups returned by the LDAP server in `Users.MemberAttribute`. Depending on the LDAP server, the attribute supports or does not support nested groups.

To determine group membership, you need to specify a value for `Groups.MemberAttribute` or for `Users.MemberAttribute`. If you specify values for both keys, both values are used, and the combination of the two group memberships is taken.

Important When creating the LDAP properties file, you need to explicitly escape certain characters in LDAP distinguished names and filter expressions. For information, see the LDAP technical specifications RFC 4515 and RFC 4514 available on the Internet.

Note If you use the Active Directory LDAP service, you may receive an error code 52e, which denotes invalid credentials. To solve the issue, ensure that you specify the correct distinguished user name and password in the `Prebind.User` and `Prebind.Password` properties.

Syntax of the LDAP properties file

Consider the following syntax rules when creating the LDAP properties file:

- A line that starts with a hash (#) is interpreted as a comment and ignored.
- Empty lines are ignored.
- All other lines must contain at least one equal sign (=). The text before the first equal sign is considered a key, and all text after the first equal sign is considered its value.
  - Both the key and value are automatically trimmed before they are interpreted.
  - You can enclose values in double quotation marks ("), in which case the text between the quotation marks is taken as the value.
- If you omit a key from the properties file, it has the same semantics as when you specify an empty value for a key. In both cases, the key is assumed to have no value.

Replace an existing contract by a new contract

During a standard installation of KCM, local instances are associated with standard contracts. You can replace standard contracts with your own custom contracts so that you can make changes that are not lost during an upgrade.

Note

- After an upgrade, if new contract types are introduced, they are not added to the contract KCM Local automatically. You need to add new contract types manually, using the `/AddContractTypeToContract` command. See Add a new contract type to a contract.
- When an instance is disassociated from a contract or deregistered and registered again, the content of the KCM Repository database is not cleared. If you do not want to share the KCM Repository content or your old contract with the new contract, you have to clear the database.

To replace contract KCM Local associated with instance 1 with a custom contract, you can remove the instance association of instance 1 from the contract KCM Local.

1. Check the instance associated with contract KCM Local.
   - Verify that instance 1 is connected to contract KCM Local using the following example command.

     ```
     ManageCM /ListContract /Partner=CCM /Customer=Local
     ```

     This command also lists the instances associated with the contract. The line `Instances associated with this Contract:` gives you information on the host and instance number associated with this contract.

2. Remove the instance association from the contract.
   - You need to remove the association between the contract KCM Local and instance 1. If a standard contract is active, use the `/Force` flag to force a breaking update. Ensure that the contract you are removing the associations from is **not** in use.

     To remove the association, execute the following example command.

     ```
     ManageCM /RemoveInstanceAssociation /Partner=CCM /Customer=Local /Force
     ```

     This command disassociates instance 1 from the contract KCM Local.
3. Remove the instance and register it again.

   After you have disassociated the instance from the contract, the instance is not yet available for a new contract. To make the instance available, remove it from the list of instances registered in the Contract Manager and register it again using the following example commands, one after the other.

   ```
   ManageCM /RemoveInstance /Host=host42 /Instance=1
   ```

   ```
   ManageCM /RegisterInstance /Host=host42 /Instance=1
   ```
4. Now you can add a new contract for this instance. For information on how to do that, see Add a new contract to the Contract Manager.

Add a new contract to the Contract Manager

Use a contract to access KCM instances with the Contract Manager. Add new contracts to the Contract Manager using the existing contract types.

Tip To skip the step-by-step procedure described below, create a contract, add a contract type, and associate an instance with the contract in one command. Skip Steps 1-10 and execute the following command, substituting the example parameters for your actual parameters.

```
ManageCM /CreateContract …
             /Partner=Northwind /Customer=Futterkiste
             /Name=CCMInteractive /Version=2
             /Host=host42 /Instance=5
```

This command adds a contract that has one contract type named `CCMInteractive` of the version `2` associated with the instance `5` on the host machine named `host42`.

1. Specify the partner. Use an existing partner or add a new partner. When you use the KCM partner in your contracts, it is considered a standard contract. Changes to this contract are lost during upgrades.
   - To get a list of the existing partners and customers associated with these partners, execute the following command, substituting the example parameters for your actual parameters.

     ```
     ManageCM /ListPartner
     ```
   - When creating a new partner name, keep in mind that the following prefixes are reserved for the Kofax Communications Manager product and cannot be used: KTA, CCM and KCM. To create a

new partner, execute the following command, substituting the example parameters for your actual parameters.

```
ManageCM /CreatePartner
```

Note One part of a contract is the partner ID. Before you create a contract, make sure that the partner ID is present in the Contract Manager. The other part of a contract is the customer ID. The combination of the partner ID and the customer ID must be unique.

2. To create a contract, execute the following command, substituting the example parameters for your actual parameters. When you use the KCM partner in your contract, it is considered a standard contract. This contract is skipped during upgrades.

```
ManageCM /CreateContract
             /Partner=Northwind /Customer=Futterkiste
```

3. Specify the contract types. A contract must have at least one contract type. The contract types must be already configured in the Contract Manager. To get a list of the available contract types, execute the following command, substituting the example parameters for your actual parameters.

```
ManageCM /ListContractType
```

4. Add one or multiple contract types to your contract. You can add both default contract types and your own contract types. In the following command, `MyContractType` is the contract type created by the user.

```
ManageCM /AddContractTypeToContract /Partner=Northwind /Customer=Futterkiste /
Name=MyContractType /Version=1
```

Also, you can add a contract and add a contract type to the Contract Manager in one command. To do so, execute the following command, substituting the example parameters for your actual parameters.

```
ManageCM /CreateContract
             /Partner=Northwind /Customer=Futterkiste
             /Name=MyContraxtType /Version=1
```

5. Specify the contract settings for the redirect URL.

The redirect URL is used for the KCM ComposerUI J2EE interfaces to indicate that information has been provided by the customer. When it is provided, the customer is directed to the redirect URL. If you do not have contract types of the type *composeruij2ee*, the value of the redirect URL is not applicable.

6. Specify the contract settings for the Export setting.

You can activate specific export functionality of KCM Designer, if needed. This export functionality is aimed to create an export of a reference KCM Designer project and store it at a location configured for the Contract Manager. To activate the functionality, set Export to true. The reference project functionality exceeds the scope of this guide. To learn more on this functionality, contact your KCM sales representative.

7. Specify the contract preferences for the Import setting.

You can activate specific import functionality of KCM Designer. This import functionality is aimed to import a reference KCM Designer project that was exported using the Export functionality described in Step 6.

Note To use this Export/Import functionality, ensure that both contracts have the same partner ID and are registered with the same Contract Manager.

8. Register unregistered instances. To get a list of the existing instances, execute the following command, substituting the example parameters for your actual parameters.

   ```
   ManageCM /ListInstance /Available
   ```

   To get a list of the contract types that are supported by the instance, execute this command, substituting the example parameters for your actual parameters.

   ```
   ManageCM /ListInstance /Host=host42 /instance=5
   ```

   - If you have a setup with a remote Contract Manager, a new instance is not registered automatically. To register an unregistered instance, execute the following command, substituting the example parameters for your actual parameters.

     ```
     ManageCM /RegisterInstance /Host=host42 /Instance=5
     ```

   - If you have the Contract Manager and an instance installed on the same computer, the instance is registered automatically, but you need to update the registration in the event of changes. To do so, execute the following command, substituting the example parameters for your actual parameters.

     ```
     ManageCM /RegisterInstance /Host=host42 /Instance=5 /update
     ```

     This command also gives information on mismatches, if any, between interfaces and interface implementation that prevent contract types from becoming available for an instance.

9. Notify the Contract Manager to reload its configuration. To do so, execute the following command.

   ```
   ManageCM /ReloadConfiguration
   ```

10. You can now associate the Contract Manager with the instance. To do so, execute the following command, substituting the example parameters for your actual parameters.

    ```
    ManageCM /AssociateInstance … /Partner=Northwind /Customer=Futterkiste
              /Host=host42 /Instance=5
    ```

    You can also activate your instance in this step. To do so, add the `/Activate` flag as shown in the example command below and skip Step 11.

    ```
    ManageCM /AssociateInstance /Partner=Northwind /Customer=Futterkiste
    /Host=host41 /Instance=5 /activate
    ```

11. You are now ready to activate your contract. To do so, execute the following command, substituting the example parameters for your actual parameters.

    ```
    ManageCM /ActivateInstance /Partner=Northwind /Customer=Futterkiste
              /Host=host42 /Instance=5
    ```

12. To continue, proceed with Add a new contract type to the Contract Manager.

Add a new contract type to the Contract Manager

You can create new contract types using the existing interfaces. To add new contract types to the Contract Manager, proceed with the following steps. To update an existing contract type, see Update an existing contract type.

1. Select interfaces to add to the contract type.
   - To get a list of the existing interfaces, execute the following command, substituting the example parameters for your actual parameters.

     ```
     ManageCM /ListInterface
     ```

2. Add the interfaces that you selected to the <interface> node of the contract type XML.

   For each <interface> node, add the <name> node and the <type> node identifying the interface to add, as shown in this example.

```
<?xml version="1.0" encoding="UTF-8"?>
<contracttypes formatversion="1">
  <contracttype>
    <name>MyContractType</name>
    <version>1</version>
    <type>core</type>
    <interfaces>
      <interface>
        <name>MyOwnInterface</name>
        <version>1</version>
      </interface>
    </interfaces>
  </contracttype>
</contracttypes>
```

3. Specify the type of your contract type.

   If you only add interfaces of the type *core*, your contract type will be *core* as well. If you add one or multiple interfaces of either the type *composeruihtml5* or *composeruij2ee*, you contract type will either be *composeruihtml5* or *composeruij2ee*.

```
<contracttype>
    <name>MyContractType</name>
    <version>1</version>
    <type>core</type>
    <interfaces>
      <interface>
        <name>MyOwnInterface</name>
        <version>1</version>
      </interface>
    </interfaces>
  </contracttype>
```

4. Give a name and a version to your contract type in the <name> and <version> nodes of the contract type XML.

   To create a new version of an existing contract type, use the name of this existing contract type, but specify a new version number. The combination of the name and the version must be unique. You cannot use the names of the default contract types. When creating a new contract type name, keep in mind that the following prefixes are reserved for the Kofax Communications Manager product and cannot be used: KTACCM, KTAKCM, CCM and KCM.

```
  <contracttype>
    <name>MyContractType</name>
    <version>1</version>
    <type>core</type>
    <interfaces>
      <interface>
        <name>MyOwnInterface</name>
        <version>1</version>
      </interface>
    </interfaces>
  </contracttype>
```

5. Add the contract type to the Contract Manager. To do so, execute the following command, substituting the example parameters for your actual parameters.

```
ManageCM /AddContractType
             /Definition=C:\Configuration\Types.xml
```

6. Update the instance registration. To do so, execute the following command, substituting the example parameters for your actual parameters. This command also shows whether the instance supports all contract types.

   ```
   ManageCM /RegisterInstance /Host=host42 /Instance=1 /Update
   ```

   If the registration is successful, you receive a list of contract types that are available for this instance. If the registration is unsuccessful, the command returns a list of errors that prevent the contract type from becoming available.

7. Reload the Contract Manager configuration using the following command.

   ```
   ManageCM /ReloadConfiguration
   ```

8. To check that the changes are available, get the WSDL. Use the following link changing the name and the version to those of your contract type.

   http://<machine_name>:8081/ccm/wsdl?contracttypename=MyContractType&contracttypeversion=V1

9. To use your contract type, you need to add it to a contract. See Add a new contract type to a contract.
   - To continue, proceed with Add a new interface to the Contract Manager.

Update an existing contract type

You can add new interfaces to an existing contract type that you created and added to the Contract Manager. You cannot add new interfaces to the default contract types. To add a new interface and update the contract type accordingly, perform the following steps.

1. To add a new interface, in the contract type XML, list all the interfaces already existing in the contract type and then add the new interface, as shown in the following example.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <contracttypes formatversion="1">
     <contracttype>
       <name>MyContractType</name>
       <version>1</version>
       <type>core</type>
       <interfaces>
         <interface>
           <name>PreviousInterface</name>
           <version>1</version>
         </interface>
         <interface>
           <name>MyOwnInterface</name>
           <version>1</version>
         </interface>
       </interfaces>
     </contracttype>
   </contracttypes>
   ```

2. Check the type of the contract type.

   If the interface you are adding is of the types *composeruithtml5* or *composeruij2ee* and the type of the contract type is of the type *core*, you need to change the type of the contract type. The only allowed direction of change is from *core* to *composeruihtml5* or to *composeruij2ee*; otherwise, you need to remove interfaces with the `/Force` flag.

3. Update the contract type in the Contract Manager.

   ```
   ManageCM /AddContractType
               /Definition=C:\Configuration\Types.xml /update
   ```

4. Register the instance. When the instance is already registered, you need to add the `/Update` flag to the command. Execute the following command, substituting the example parameters for your actual parameters.

```
ManageCM /RegisterInstance /Host=host42 /Instance=2 /Update
```

Note Once an instance is registered, the name of the host is fixed for this instance. It does not map localhost to the name of your machine. Make sure to register instances to the name of your computer.

5. Reload the Contract Manager configuration using the following command.

```
ManageCM /ReloadConfiguration
```

6. To check that the changes are available, get the WSDL. Use the following link, changing the name and version to those in your contract type.

   http://<machine_name>:8081/ccm/wsdl?contracttypename=MyContractType&contracttypeversion=V1

Add a new interface to the Contract Manager

When the existing interfaces are not sufficient, you can add new ones.

1. Specify a KCM Core service script that will implement the new interface.

   Add the <corescript> node to the interface XML and enter the name of the script as shown in the following example. The script name is the name without the .dss extension.

```
<interface uselegacysessionid="false" session="none">
  <name>MyOwnInterface</name>
  <version>1</version>
  <type>core</type>
  <corescript>MyOwnScript</corescript>
  <parameters>
    <input>
      <param optional="false">
        <name>textIn</name>
        <type>message</type>
      </param>
    </input>
    <output>
      <param optional="false">
        <name>textOut</name>
        <type>message</type>
      </param>
    </output>
  </parameters>
</interface>
```

   To add a new service script to KCM Core, use KCM Core Administrator. For more information, see the *Kofax Communications Manager Core Developer's Guide*. Also, you can add a script to the KCM Core library. For more information, see the *Kofax Communications Manager Core Scripting Language Developer's Guide*.

2. Specify the interface parameters.

   Parameters are either input parameters or output parameters. Input parameters can be optional.

   - **Input parameters** are parameters that you provide. Your KCM Core script must contain the same parameters as the interface. The only exception is parameters of the type *document*. If the interface contains such parameters, you do not need to specify them for the KCM Core script.

     To add an input parameter, add a <parameter> node, <name> and <type> subnodes to the XML file.

     ```
     <interface uselegacysessionid=”false” session=”none”>
       <name>MyOwnInterface</name>
       <version>1</version>
       <type>core</type>
       <corescript>MyOwnScript</corescript>
       <parameters>
         <input>
           <param optional=”false”>
             <name>textIn</name>
             <type>message</type>
           </param>
         </input>
         <output>
           <param optional=”false”>
             <name>textOut</name>
             <type>message</type>
           </param>
         </output>
       </parameters>
     </interface>
     ```

     These parameters appear as parameters in your interface request SOAP calls. The parameter `textIN` appears in your KCM Core script as parameter text `textIN`.

   - *Optional parameters* are parameters that may be omitted from the SOAP request.

     ```
         <input>
            <param optional=”true”>
              <name>textIn</name>
              <type>message</type>
            </param>
          </input>
     ```

     If optional is set to true, the KCM Core script must contain a default value for the parameter that may appear in the following way: `parameter text textIN = "Optional value"` or `parameter text textIN = ""`.

     Parameters can be various types. The types differ according to restrictions on the values that can be entered. The following table provides a description of the input parameter types used in the KCM Core script.

     Note The input parameter types are case-sensitive and must be all lowercase. The parameter names are case-insensitive.

Input parameter type	Input parameter value
boolean	Can only be true or false. It should match with a boolean parameter in the KCM Core script.
message	Must contain text

Input parameter type	Input parameter value
name	Can only contain alphanumeric characters A-Z, a-z, 0-9
repositoryobjectname	Can only contain those characters that are allowed for repository object, excluding the following characters: ^:?/<>*"\()
url	Any URL
document	A file to be retrieved using RetrieveFile in the KCM Core script

See the following example.

```
      <input>
        <param optional="false">
          <name>my_xml</name>
          <type>document</type>
        </param>
```

The preceding example corresponds to the following in the KCM Core script:

```
 Const Text my_xml = "inputxml"[_SessionDir,"xml"];
ReceiveFile
    Src    ("inputxml")
    Dest (my_xml);
```

This retrieves the value of the input parameter to the file inputxml.xml stored in the session directory for this script execution.

- **Output parameters** are parameters returned by the KCM Core script. To retrieve output parameters, you can use `exchange_data` or SendFile when a parameter with type *document* is concerned. See the following example.

```
      <output>
        <param optional="false">
          <name>textOut</name>
          <type>message</type>
        </param>
      </output>
```

The preceding example corresponds to the following in the KCM Core script:

```
Var Text outputValue = "output parameter value";
Var Text result;
Result = exchange_data("textOut", outputValue, 0);
```

The `outputValue` contains the value to return.

The following table provides a description of the output parameters types used in the KCM Core script.

Important The output parameter types are case-sensitive and must be all lowercase. The parameter names are case-insensitive, **except for** the name of a parameter with type document, which must also be lowercase.

Output parameter type	Output parameter value
message	Must contain text

Output parameter type	Output parameter value
interactiveresult	The output is either "document" or "session." This indicates whether the result of an interactive run was a document or a session. A session is returned when an interactive run is suspended.
url	Any URL
document	A file to be sent using SendFile in the KCM Core script

The parameter with type *document* is a special parameter type. It should be returned using SendFile with the parameter call as the destination. Optional parameters may not always be returned.

An example is provided here.

```
      <output>
        <param optional=”false”>
          <name>return_document</name>
          <type>document</type>
        </param>
```

The preceding example corresponds to the following in the KCM Core Script:

```
 Const Text return_document = “mydocument”[_SessionDir,”docx”];
SendFile
    Dest   (“return_document”)
    Src (return_document);
```

This returns the value for the output parameter `return_document`. The file returned is return_document.docx stored in the session directory for this script execution.

Note Like all other output parameters, the parameter with type *document* must be lowercase. If it is not lowercase, an error message indicating the issues appears.

3. Specify the type of the interface.

   You can indicate if your task should be handled by KCM Core only, or if the task involves ComposerUI for HTML5 or ComposerUI for J2EE. ComposerUI for HTML5 or for J2EE are necessary to run KCM Core scripts that involve interactivity. This is necessary to compose documents that contain forms or Content Wizards with selections to be made.

   Add the <type> node and enter the type as shown in this example.

```
<interface uselegacysessionid="false" session="none">
    <name>MyOwnInterface</name>
    <version>1</version>
   <type>core</type>
    <corescript>MyOwnScript</corescript>
    <parameters>
      <input>
        <param optional="false">
          <name>textIn</name>
          <type>message</type>
        </param>
      </input>
      <output>
        <param optional="false">
          <name>textOut</name>
          <type>message</type>
        </param>
```

```
      </output>
    </parameters>
  </interface>
```

4. Specify a name and a version number for the interface.

   If you create a variant of an existing script, you can give it the same name but add a new version. The name and version combination must be unique. See the following example.

```
  <interface uselegacysessionid="false" session="none">
    <name>MyOwnInterface</name>
    <version>1</version>
    <type>core</type>
    <corescript>MyOwnScript</corescript>
    <parameters>
      <input>
        <param optional="false">
          <name>textIn</name>
          <type>message</type>
        </param>
      </input>
      <output>
        <param optional="false">
          <name>textOut</name>
          <type>message</type>
        </param>
      </output>
    </parameters>
  </interface>
```

5. Use a legacy session ID.

   Note The value for `uselegacysessionid` must always be false. New scripts do not need to have `uselegacysessionid` set to true.

```
<?xml version="1.0" encoding="UTF-8"?>
<interfaces formatversion="1">
  <interface uselegacysessionid="false" session="none">
    <name>MyOwnInterface</name>
```

6. Specify the session settings.

```
<?xml version="1.0" encoding="UTF-8"?>
<interfaces formatversion="1">
  <interface uselegacysessionid="false" session="none">
    <name>MyOwnInterface</name>
```

   Indicate if the script needs to create a session in the Contract Manager (`session="create"`) or requires a session to be present (`session="require"`) in the Contract Manager. You can use this functionality to link various interfaces together. For example, if you have one interface that creates a document but does not return the document, the script creates a session, and the document is stored

in that session. The other interface that required a session distributes the document stored in this session to a location.

If you have your type set to *composeruihtml5* or *composeruij2ee*, the create session will be implicit. Your session should be set to *none* as this attribute is ignored. For interfaces of the type core, the session attribute can be set to *none*, *create*, or *require*.

- To link various interfaces, you add one interface that creates a session in the Contract Manager. The call to this first interface creates a session. Then create other interfaces that require a session. When calling these interfaces, enter the session retrieved from the first call.

  If you set the session to **"create**,**"** your interface automatically returns a session ID. You can use this session ID with other requests that require a session ID.

  A KCM Core script added to an interface that creates a session in the Contract Manager should always create a session and pass it on when the script is executed.

  Note The session ID generated in the KCM Core script is linked to, but not equal to, the session ID generated in the Contract Manager. The session ID returned by the SOAP request is the Contract Manager session ID.

  The following are examples of what the KCM Core script should contain. Both examples perform the same functionality.

  ```
  CreateSession;
  Const Text dummy = exchange_data("sessionID", _sessionid, 0);
  ```

  or

  ```
  ITPOLSSessionStart
  ```

- To use an interface with the session set to **"require**,**"** perform a request that first creates a session and then receives a session ID from it. Interfaces with the session set to "require" are automatically extended with a parameter to enter the session ID. This allows you to have access to a particular session in KCM Core. You can access data stored in the KCM Core session using `_sessiondir`.

7. Write the collected information to an interface XML file. The content of the file should look similar the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<interfaces formatversion="1">
  <interface uselegacysessionid="false" session="none">
    <name>MyOwnInterface</name>
    <version>1</version>
    <type>core</type>
    <corescript>MyOwnScript</corescript>
    <parameters>
      <input>
        <param optional="false">
          <name>textIn</name>
          <type>message</type>
        </param>
      </input>
      <output>
        <param optional="false">
          <name>textOut</name>
          <type>message</type>
        </param>
      </output>
    </parameters>
  </interface>
</interfaces>
```

8. Add the interface to the Contract Manager.

   Use the ManageCM tool to add the interface to the Contract Manager. Use the following command, substituting the example parameters for your actual parameters.

   ```
   ManageCM /AddInterface
               /Definition=C:\Configuration\myinterface.xml
   ```

   To use this interface, add it to a contract type.

9. Reload the configuration using the following command.

   ```
   ManageCM /ReloadConfiguration
   ```

10. To continue, proceed with Create and deploy interface implementation.

Create and deploy interface implementation

When you add new interfaces, you need to add new interface implementations.

KCM offers contracts for KCM users to access KCM functionality such as composing a .docx document or composing a Document Pack. A contract can contain one or multiple contract types. Example standard contract types are CCMInteractiveV2 or CCMDistributionV1. A contract type contains a number of interfaces to access the KCM functionality. Example standard interfaces are ComposeDocxV1 or ComposeDocumentPackV1. You can add new interfaces to new contract types.

Most interfaces are linked to a KCM Core interface script that implements the functionality to be performed when accessing the interface. The interface script must be added to the KCM Core library CDS files so it becomes available for KCM.

To create and deploy an interface implementation, you need to have the following:

- The **Script.exe** creating tool to create a KCM Core script library.

  1. To locate the tool, navigate to:

  `<deploy root>\KCM\Programs\<KCM version>\ITP Server\bin`

  On the list with installed executables, find script.exe.

  2. Copy script.exe to your desired destination. **Example** `C:\CreateLibrary`

- **Scripts to add**. Create a folder for the scripts to add. **Example** `C:\CreateLibrary\Scripts`

  You should store scripts as .dss files. The name of the *.dss file must match that of the script. The content follows the rules of the Core scripting language as described in the *Kofax Communications Manager Core Scripting Language Manual*.

- **Exit point scripts**. Exit point scripts must be stored as .dss files. The name of the *.dss file must match that of the exit point. After you have deployed the library, all the exit point scripts must be available when you recompile your scripts.

  Note When you make changes to exit points, make sure to recompile your script for the changes to take effect.

- **SPEC file**. The SPEC file contains the following specifications needed to create a library file.

  A list with constants available for use in scripts

  A list with internals (scripts) to expose as available services

  A list with exit points (optional)

  Maximum log level for the library

  Company (required)

  Description

  MinVersion

  Library (required)

  ScriptPath (required)

For more information, see the example SPEC file that resides in: `<deploy root>\KCM\Documentation\<KCM version>\Resources\Examples\Core Scripting`

- **KCM Core script library (itpserver.cds)**

  1. Locate the KCM Core script library for the instance. All instances contain the same KCM Core script library so you do not need to repeat this for each instance. You can copy the itpserver.cds file to a new location, if needed.

  2. To locate itpserver.cds, navigate to:

  `<deploy root>\KCM\Work\<KCM version>\Instance_01\core\Config`

  3. Enter the location of the file in the previously created SPEC file as library=<location>\itpserver.cds>. For more information, see the example SPEC file that resides in: `<deploy root>\KCM\Documentation\<KCM version>\Resources\Examples\Core Scripting`

Create a library

Execute the following command. `<spec>` indicates the SPEC file name that specifies the scripts to add.

```
Script.exe -lib library.cds <spec>
```

A library.cds file is created. It contains the scripts listed in the SPEC file. If the scripts contain errors, the library.cds file fails to be created. Script.exe returns error messages indicating the issues it encountered.

- To create an additional library, execute the following command.

```
Script.exe -lib <name>.cds <spec>
```

Deploy a library

1. Substitute the library.cds file (see Create a library) for the library.cds file located at the instance to which you want to deploy.

   Tip Make a backup of the original library.cds file before you replace this file. This file can be used to return to the original situation, if necessary.

   To locate the library.cds file, navigate to:

   `<deploy root>\KCM\Work\<KCM version>\<Instance_#>\core\Config`

   where `<KCM version>` is the installed KCM version and `<Instance_#>` is the instance to which you want to deploy the library.cds file.
2. Restart all Document Processors for the library to become active.

   To restart the document processors, you can use KCM Core Administrator or the Windows Services control panel applet.

Deploy an additional library

1. Place the library in the same directory as the library.cds file located at the instance to which you want to deploy. Also, see Create a library.

   To locate the library.cds file, navigate to `<deploy root>\KCM\Work\<KCM version>\<Instance_#>\core\Config`, where `<KCM version>` is the installed KCM version and `<Instance_#>` is the instance to which you want to deploy the cds file.

   Tip If you are replacing an existing .cds file, make a backup of the original file before you replace it. You can later use this file to return to the original situation, if necessary.

2. Register the new library.

   Before proceeding, make sure that no KCM Core Administrator is open for the instance that you want to register the new library for.

   Open the dp.ini file for that instance and add the following line to the configuration section.

   ```
   LibraryList = <name>.cds
   ```

   - To register multiple libraries, list them using a semicolon as a separator, as shown below.

     ```
     LibraryList = <name1>.cds;<name2>.cds
     ```

     To locate the dp.ini file, navigate to:

     `<deploy root>\KCM\Work\<KCM version>\<Instance_#>\core\Config`

     where `<KCM version>` is the installed KCM version and `<Instance_#>` is the instance to which you want to deploy the CDS file.

     Note You can register at most 30 additional libraries.

3. Restart all Document Processors for the library to become active.
   To restart the document processors, you can use KCM Core Administrator or the Windows Services control panel applet.

Manage calling applications and their access to Contract Manager interfaces

If the `ContractManager!UseAuthentication` deployment parameter is set to True, when a calling application executes SOAP calls on the Contract Manager, it must supply the application name and the key.

This mechanism uses HTTP basic authentication. This means that you should only use authentication and authorization in combination with an encrypted connection. KCM expects preemptive authentication.

You can manage calling applications using the ManageCM tool.

1. Add an application using the `/AddApplication` command.
   This command adds the application and shows the key that needs to be passed when calling the interfaces. For the usage examples and information on the other available commands, see Add an application, List applications, Get a new key for the application, and Remove an application.
2. In order for the changes to take effect, use the `/ReloadConfiguration` command.
3. When an application is successfully authenticated when performing a call to a Contract Manager interface, it is checked whether it has a privilege to call this interface. A privilege is a permission to use interfaces in the Contract Manager.
   A privilege can take four forms:
   `<Partner>`: The application can call all interfaces for the designated partner.
   `<Partner>` and `<Customer>`: The application can call all interfaces for the designated customer, but only for the designated partner.
   `<Partner>`, `<Customer>`, and `<Contract type (name and version)>`: The application can call all interfaces for the designated contract, but only for the designated partner and customer.
   `<Partner>`, `<Customer>`, `<Contract type (name and version)>`, and `<Interface (name and version)>`: The application can call the designated interface, but only for the designated partner, customer, and contract.
4. Add a privilege for the calling application using the `/AddPrivilegeForApplication` command.

   Note You need to add at least one privilege for the application before it can access the Contract Manager interfaces.

   For the usage examples and information on the other available commands, see Add a privilege for the application, List privileges for the application, Get all privileges in an XML file, Set privileges, and Remove a privilege for the application.
5. Execute the`/ReloadConfiguration` command to apply the configuration changes.
   - You can export applications with the `/ExportDatabase` command. For more information, see Export the Contract Manager internal database to a file.

Chapter 3

Use the ManageCM tool

This chapter describes the ManageCM tool functionality.

The ManageCM tool is designed to manage the KCM Contract Manager configuration. The configuration is stored in an internal database. Use the ManageCM tool to define contract types, manage contracts, control instances as well as associate instances with contracts and activate them.

Note Changes you make to standard contracts, contract types, or interfaces are lost during upgrades. Standard contracts are those contracts with a partner that can be either KCM or CCM.

The ManageCM tool resides in:

`<deploy root>\KCM\Programs\<version>\Management`

The `ManageCM` command line has the following structure.

```
ManageCM <cmd> <par>
```

`cmd`: A flag that defines a command to be performed. A flag is used to enable features.

It is written as `/flag`.

`par`: Zero, one, or multiple parameters and flags that apply to the `ManageCM` command. A parameter is written as `/parameter=value` to define a value of the parameter. If the value contains spaces, the parameter should be enclosed in quotes.

Note Flags and parameters are case-insensitive.

Example

```
ManageCM /CreatePartner
            /Partner=Example
```

Also, the ManageCM tool provides short help on the command line with the `/?` parameter.

- `ManageCM /?` shows the available commands.
- `ManageCM /command /?` provides help about the `/command` command.

Note When using the ManageCM tool from the command prompt, start the command prompt with elevated rights. When using the ManageCM tool to add interfaces, your account needs to have the Modify permission on the Config folder that resides in: `<deploy root>\KCM\Work\<version>\ContractManager\Config`

Consistency checks

The `ManageCM` command performs consistency checks prior to performing a requested operation. If the requested operation leads to a breakdown in the functionality, the `ManageCM` command fails.

Some commands provide the `/Force` flag that you can use to override particular consistency checks. Use the `/Force` flag during development to make configuration changes that may result in a breakdown. You must not use the `/Force` flag in the production environment.

Note Errors lead to a complete rollback of the changes made to the KCM Contract Manager, unless specified otherwise.

Activate changes

The `ManageCM` command makes changes to the configuration stored in the Contract Manager internal database. These changes are not activated until the configuration is reloaded, unless specified otherwise.

Use the `/ReloadConfiguration` command to apply configuration changes. This command works only from the server hosting the Contract Manager service.

If the command option is not available, you may force a reload of the configuration by either restarting the service "Apache Tomcat CCMRuntime instance <version number>" or visiting the following URL:

- http://localhost:8081/ccm/Administration/ReloadDatabase

Note You can access the URLs only from the server hosting the Contract Manager Service. Also, reloading the configuration does not interrupt active sessions.

Commands

This section lists and describes the ManageCM tool commands.

This table lists the commands, gives a short description, and provides links to the corresponding sections.

Parameter	Usage	Link to the Section
`/ReloadConfiguration`	Applies any configuration changes made.	Reload the configuration
`/CreatePartner`	Creates a new partner.	Create a partner
`/RemovePartner`	Removes an existing partner.	Remove a partner
`/ListPartner`	Lists partners and customers.	Show information about partners
`/CreateContract`	Creates a contract and optionally instantiates it.	Create a contract

Parameter	Usage	Link to the Section
/ListContract	Lists contracts and detail information.	Show a contract
/ChangeContract	Changes the properties of a contract.	Change properties of a contract
/RemoveContract	Removes an existing contract.	Remove a contract
/AddContractType	Creates a contract type.	Create contract types
/ListContractType	Lists contract types and detail information.	Show contract types
/RemoveContractType	Removes a contract type.	Remove contract types
/AddInterface	Adds an interface.	Add interfaces
/ListInterface	Lists interfaces and detail information.	List interfaces
/RemoveInterface	Removes an interface.	Remove interfaces
/AddApplication	Adds an application and prints the key that needs to be passed when calling a Contract Manager interface.	Add an application
/ListApplications	Lists applications known to the Contract Manager.	List applications
/GetNewKeyForApplication	Generates a new key for the application and invalidates the older keys.	Get a new key for the application
/RemoveApplication	Removes the application and its associated privileges.	Remove an application
/AddPrivilegeForApplication	Adds privileges for the application.	Add a privilege for the application
/ListPrivilegesForApplication	Lists the privileges for the application.	List privileges for the application
/GetPrivileges	Returns all privileges in a privileges XML file.	Get all privileges in an XML file
/SetPrivileges	Sets several privileges at the same time using a privileges XML file.	Set privileges
/RemovePrivilegeForApplication	Removes privileges for the application.	Remove a privilege for the application
/AddContractTypeToContract	Adds a contract type to an existing contract.	Add a new contract type to a contract
/RemoveContractTypeFromContract	Removes a contract type from an existing contract.	Remove a contract type from a contract
/RegisterInstance	Adds an instance to the Contract Manager.	Register or update one or multiple instances

Parameter	Usage	Link to the Section
`/ListInstance`	Lists instances and detail information.	Show instances
`/ChangeInstance`	Changes the attributes of an instance.	Change properties of an instance
`/RemoveInstance`	Deregisters an instance from the Contract Manager.	Remove the registration of an instance
`/AssociateInstance`	Associates an instance with a contract.	Associate an instance with a contract
`/RemoveInstanceAssociation`	Removes instance associations from a contract.	Remove instance associations from a contract
`/ActivateInstance`	Activates a previously assigned instance on a contract.	Switch processing of a contract to the latest associated instance
`/RollBackInstance`	Switches a contract back to the previous instance it used.	Switch processing of a contract to the previous instance
`/ExportDatabase`	Exports the Contract Manager configuration (stored in an internal database) to a file.	Export the Contract Manager internal database to a file
`/ImportDatabase`	Imports the Contract Manager configuration from a file to an internal database.	Import content from a file to the Contract Manager internal database
`/SetSharedResource`	Finds or changes the location of the folder for import and export shared resources	Find or change the location of the shared resources folder

Reload the configuration

The `/ReloadConfiguration` command reloads the KCM Contract Manager configuration, which applies any changes made with the other commands.

This command has no parameters and can only be used on the computer hosting the Contract Manager service.

Create a partner

The `/CreatePartner` command adds a new partner to the KCM Contract Manager configuration.

This command has the following parameter:

- `/Partner=` *Required*. Specifies the name for the new partner.

The following example demonstrates how the `Northwind` partner can be added to the configuration.

```
 ManageCM /CreatePartner …
/Partner=Northwind
```

Remove a partner

The `/RemovePartner` command removes a partner from the KCM Contract Manager configuration.

This command has the following parameter:

- `/Partner=` *Required*. Specifies the name of the partner to be removed.

The command fails if the partner has contracts.

The following example demonstrates how the `Northwind` partner can be removed from the configuration.

```
 ManageCM /RemovePartner …
/Partner=Northwind
```

Show information about partners

The `/ListPartner` command lists partners and customers/contracts belonging to these partners.

This command has the following parameter:

- `/Partner=` *Optional*. Shows information about this partner.

Use the `/Partner=` parameter to get a list of all customers and contracts for this partner. Omit the parameter to get a list of all partners with their customers and contracts.

The following example demonstrates how all contracts belonging to the `Northwind` partner can be listed.

```
ManageCM /ListPartner … /Partner=Northwind
```

Create a contract

The `/CreateContract` command creates a new customer with a new contract. You can optionally populate this contract with a contract type and associate it with an instance.

This command has the following parameters and flags:

`/Partner=` *Required*. Specifies the partner to which this contract belongs. When partner is KCM or CCM, this is considered a standard contract. This contract is skipped during upgrades.

`/Customer=` *Required*. Specifies the new customer to be created to which this contract belongs.

`/RedirectURL=` *Optional*. Specifies a redirect URL for the contract.

`/Export` *Optional*. `/Export=Y` enables the export feature for reference projects; other values disable this feature.

`/Import` *Optional*. `/Import=Y` enables the import feature for reference projects; other values disable this feature.

`/Name=` *Optional*. The name of the initial contract type to be assigned to this contact.

`/Version=` *Optional*. The version number of the initial contract type to be assigned to this contact.

`/Host=` *Optional*. The server hosting the instance to be associated with this contract.

`/Instance=` *Optional*. The number of the instance to be associated with this contract.

`/OutputManagementHotfolder=` *Optional.* The request folder for the Batch & Output Management component. This folder must be accessible from the server on which the instance is running.

`/Toolboxpath=` *Optional*. Specifies the location of the `ccmforword manifest-<KCM version>.xml` file for KCM Toolbox for Word. This location must accessible to all users and listed in the Trusted App Catalogs section in Microsoft Word.

To create an empty contract, you only need the `Partner/` parameter and the `/Customer` parameter.

You can add the `/Name=` and `/Version=` parameters to an initial contract type. If an initial contract type has been assigned, you can associate the contract with an available instance using the `/Host=` and `/Instance=` parameters.

If further optional steps fail, such as when the instance is being unavailable or the requested contract type cannot be provided, the contract is not created.

The following set of commands creates the customer `Futterkiste`, creates a contract under the partner `Northwind`, assigns the `CCMInteractive` version `2` contract type, and associates the instance `5` with the contract on the server `host42`.

```
ManageCM /CreateContract …
             /Partner=Northwind /Customer=Futterkiste
             /Name=CCMInteractive /Version=2
             /Host=host42 /Instance=5
```

Note A contract is unavailable for use until the instance is activated and the KCM Contract Manager is restarted or the configuration is reloaded.

Show a contract

The `/ListContract` command lists properties, assigned contract types, and associated instances.

This command has the following parameters:

- `/Partner=` *Required*. The partner of the contract.
- `/Customer=` *Required*. The customer of the contract.

The following example demonstrates how the configuration of the contract belonging to the partner `Northwind` and the customer `Futterkiste` can be shown.

```
ManageCM /ListContract … /Partner=Northwind /Customer=Futterkiste
```

Remove a contract

The `/RemoveContract` command removes a contract.

This command has the following parameters and flags:

- `/Partner=` *Required*. The partner of the contract.
- `/Customer=` *Required*. The customer of the contract.
- `/Force` *Optional*. Allows the removal of contracts with an active instance.

A contract with an active instance added to it is assumed to be in use, and it cannot be removed unless you use the `/Force` flag.

Once a contract is deleted, you can create a new contract for the same partner and customer with the `/CreateContract` command.

The following example demonstrates how the contract belonging to the partner `Northwind` and the customer `Futterkiste` can be removed.

```
ManageCM /RemoveContract … /Partner=Northwind /Customer=Futterkiste
```

Change properties of a contract

The `/ChangeContract` command changes the properties of an existing contract.

Note If you make changes to standard contracts, these changes are lost when you upgrade. Standard contracts are those contracts with a partner that can be either KCM or CCM.

This command has the following parameters:

`/Partner=` *Required*. The partner of the contract.

`/Customer=` *Required*. The customer of the contract.

`/RedirectURL=` *Optional*. New redirect URL.

`/Export=` *Optional*. `/Export=Y` enables the export feature for reference projects; other values disable this feature.

`/Import=` *Optional*. `/Import=Y` enables the import feature for reference projects; other values disable this feature.

`/OutputManagementHotfolder=` *Optional.* The request folder for the Batch & Output Management component. This folder must be accessible from the server on which the instance is running.

`/Toolboxpath=` *Optional*. Specifies the location of the `ccmforword manifest-<KCM version>.xml` file for KCM Toolbox for Word. This location must accessible to all users and listed in the Trusted App Catalogs section in Microsoft Word.

`/Force` *Optional*.

Use the `/Force` flag to change contracts that have an active instance.

At least one of the optional parameters must be present. If the parameters are omitted, values of the contract properties are not changed.

The following example demonstrates how the export feature for the contract belonging to the partner `Northwind` and the customer `Futterkiste` can be disabled.

```
ManageCM /ChangeContract … /Partner=Northwind /Customer=Futterkiste
         /Export=N
```

Create contract types

The `/AddContractType` command adds one or multiple contact types.

This command has the following parameter and flags:

`/Definition=` *Required*. An XML file containing the definition of one or multiple contract types.

`/Update` *Optional*. Allows existing contract types to be extended.

`/Force` *Optional*.

The command fails if at least one of the new contract types already exists, unless you use the `/Update` flag. Updates are permitted only if new definitions extend the existing contract types.

The `/Force` flag allows you to remove interfaces from existing contract types. This operation may break functionality and should not be performed in a production environment.

The format of the `/Definition=` XML file is described in the section Add a new contract type to the Contract Manager.

The following example demonstrates how to add the contract types from the file `C:\Configuration\Types.xml`.

```
 ManageCM /AddContractType …
/Definition=C:\Configuration\Types.xml
```

Show contract types

The `/ListContractType` command either provides detailed information on a particular contract type or shows a list containing all contract types.

This command has the following parameters:

- `/Name=` *Optional*. Name of the contract type.
- `/Version=` *Optional*. Version number of the contract type.

Use the `/Name=` and `/Version=` parameters to get information on a particular contract type, including a list of interfaces it contains. If none of the parameters is used, a list of all contract types is shown.

The following example demonstrates how a list of all contract types can be triggered.

```
ManageCM /ListContractType …
```

The second example shows how detailed information on the version `1` of the contract type `ISVPerformAction` can be shown.

```
ManageCM /ListContractType … /Name=ISVPerformAction /Version=1
```

Remove contract types

The `/RemoveContractType` command removes one or multiple contract types.

This command has the following parameters:

- `/Definition=` *Optional*. An XML file containing the definition of one or multiple contract types.
- `/Name=` *Optional*. Name of the contract type.
- `/Version=` *Optional*. Version number of the contract type.

Note You cannot remove contract types in use by a contract.

Use the `/Definition=` parameter to remove all contract types described in the XML file. This command accepts the same XML file as the `/AddContractType` uses, allowing you to create and remove contract types with the same XML file. If a contract type described in the XML file does not exist, the removal continues. If one or multiple contract types could not be removed, the command fails, and no changes are made.

Use the `/Name=` and `/Version=` parameters to remove a particular contract type.

Note You cannot use the `/Definition=` and `/Name=` parameters together in one request.

The following example demonstrates how to remove all contract types from the file `C:\Configuration\Types.xml`.

```
 ManageCM /RemoveContractType …
/Definition=C:\Configuration\Types.xml
```

The second example shows how the version `1` of the contract type `ISVPerformAction` can be removed.

```
 ManageCM /RemoveContractType …
/Name=ISVPerformAction /Version=1
```

Add interfaces

The `/AddInterface` command adds one or multiple interfaces. This command only works on the computer hosting the Contract Manager service.

This command has the following parameters and flags:

- `/Definition=` *Required*. An XML file containing a definition of one or multiple contract types.
- `/Update` *Optional*. Allows existing interfaces to be extended.
- `/Force` *Optional*.

The command fails if at least one of the new interfaces already exists, unless the `/Update` flag is used. Updates are permitted only if the new definitions are equal to or extend the existing interfaces.

Existing required parameters may be redefined as optional parameters. New optional parameters may be added at the end of the parameters list.

Other changes to existing interfaces break the interface and are rejected.

The `/Force` flag allows you to update and break existing interfaces. This operation can break functionality and must not be performed in the production environment.

The format of the `/Definition=` XML file is described in the section Add a new interface to the Contract Manager.

The following example demonstrates how to load interfaces from the file `C:\Configuration\Interfaces.xml`.

```
 ManageCM /AddInterface …
/Definition=C:\Configuration\Interfaces.xml
```

List interfaces

The `/ListInterface` command either provides detailed information on a particular interface or gives a list containing all interfaces.

This command has the following parameters:

- `/Name=` *Optional*. Name of the contract type.
- `/Version=` *Optional*. Version number of the contract type.

Use the `/Name=` and `/Version=` parameters to get detailed information on a particular interface, including information on its parameters and a list of contract types it is associated with. If none of the parameters is used, a list of all interfaces is shown.

The following example demonstrates how a list of all interfaces can be shown.

```
 ManageCM /ListInterface …
```

The second example shows how you can get detailed information on the version `1` of the interface `Perform.Action`.

```
 ManageCM /ListInterface … /Name=Perform.Action
/Version=1
```

Remove interfaces

The `/RemoveInterface` commands removes one or multiple interfaces. This command only works on the computer hosting the Contract Manager service.

This command has the following parameters:

- `/Definition=` *Optional*. An XML file containing a definition of one or multiple interfaces.
- `/Name=` *Optional*. Name of the contract type.
- `/Version=` *Optional*. Version number of the contract type.

Note You cannot remove interfaces in use by a contract type.

Use the `/Definition=` parameter to remove all interfaces described in the XML file. This command accepts the same XML file as the `/AddInterface` uses, allowing you to create and remove interfaces with the same XML file. If an interface described in the XML file does not exist, the removal continues. If one or multiple interfaces could not be removed, the command fails, and no changes are made.

Use the `/Name=` and `/Version=` parameters to remove a particular interface.

Note You cannot use the `/Definition=` and `/Name=` parameters in one request.

The following example demonstrates how to remove all interfaces from the file `C:\Configuration\Interfaces.xml`.

```
 ManageCM /RemoveInterface …
/Definition=C:\Configuration\Interfaces.xml
```

The second example shows how to remove the version `3` of the interface `ArchiveDocument`.

```
ManageCM /RemoveInterface … /Name=ArchiveDocument /Version=3
```

Add an application

The `/AddApplication` adds an application and prints the key that needs to be passed when calling a Contract Manager interface.

This command has the following parameter:

- `/Application=` *Required*. The name for the application, not exceeding 50 characters. Can only contain alphanumeric characters a-z, A-Z, 0-9 and the underscore. **Example** SampleApp_1

```
ManageCM /AddApplication
```

The following example demonstrates how to add an application called `app` and get the key fo this application.

```
ManageCM /AddApplication … /Application=app
```

List applications

The `/ListApplications` command shows a list of applications known to the Contract Manager.

The command has no parameters.

Get a new key for the application

The `/GetNewKeyForApplication` command generates a new key for the application and invalidates the older keys.

This command has one parameter:

- `/Application=`. *Required*. The name for the application.

The following example demonstrates how to get a new key for the application `app`.

```
ManageCM /GetNewKeyForApplication … /Application=app
```

Remove an application

The `/RemoveApplication` command removes the application and its associated privileges.

This command has the following parameter:

- `/Application=` *Required*. The name for the application.

The following example demonstrates how to remove the application `app` and all privileges associated with this application.

```
ManageCM /RemoveApplication … /Application=app
```

Add a privilege for the application

The `/AddPrivilegeForApplication` command adds privileges for the application.

The command has the following parameters:

- `/Application=` *Required*. The name for the application.
- `/Partner=` *Required*. The partner for which this application is privileged to call a Contract Manager interface.
- `/Customer=` *Optional*. The customer for which this application is privileged to call a Contract Manager interface.
- `/ContractTypeName=` *Optional*. The contract type name for which this application is privileged to call a Contract Manager interface.
- `/ContractTypeVersion=`. *Optional*. The contract type version for which this application is privileged to call a Contract Manager interface.
- `/InterfaceName=`. *Optional*. The interface name for which this application is privileged to call a Contract Manager interface.
- `/InterfaceVersion=`. *Optional*. The interface version for which this application is privileged to call a Contract Manager interface.

The following example shows how to add a privilege for the application `app` when the partner is `CCM`, the customer is `local`, and the contract type is `CCMInteractive` version `2`.

```
ManageCM /AddPrivilegeForApplication … /Application=app /Partner=CCM /Customer=local
 /ContractTypeName=CCMInteractive /ContractTypeVersion=2
```

List privileges for the application

The `/ListPrivilegesForApplication` command lists the privileges for the application.

The command has the following parameter:

- `/Application=` *Required*. The name for the application.

The command triggers a list of privileges for the application that you specified. The value `*all` indicates that the application can access all calls.

Get all privileges in an XML file

The `/GetPrivileges` command returns all privileges in an XML file that can be used in the `/SetPrivileges` command.

The command has one parameter:

- `/PrivilegeXML=` *Required*. The location of an XML file to save privileges to.

The following example shows how to obtain a privileges XML file that resides in `C:\Temp\privileges.xml`.

```
ManageCM /GetPrivileges … /PrivilegeXML=C:\Temp\privileges.xml
```

The following is an example privileges XML file.

```
<privilegeXML>
 <application id="ManagementApplication">
  <privilege partner="CCM"/>
 </application>
 <application id="LocalApplication">
  <privilege partner="CCM" customer="Local"/>
 </application>
 <application id="InteractiveIntegration">
  <privilege partner="CCM" customer="Local" contracttypename="CCMInteractive"
 contracttypeversion="1"/>
  <privilege partner="CCM" customer="Local" contracttypename="CCMInteractive"
 contracttypeversion="2"/>
  <privilege partner="CCM" customer="Local" contracttypename="CCMInteractive"
 contracttypeversion="3"/>
 </application>
 <application id="FreeInteractiveIntegration">
  <privilege partner="CCM" customer="Local" contracttypename="CCMInteractive"
 contracttypeversion="1" interfacename="ComposeDocumentPackGet" interfaceversion="1" />
  <privilege partner="CCM" customer="Local" contracttypename="CCMInteractive"
 contracttypeversion="1" interfacename="ComposeDocumentPackInteractiveStart"
 interfaceversion="1" />
  <privilege partner="CCM" customer="Local" contracttypename="CCMInteractive"
 contracttypeversion="1" interfacename="ComposeDocumentPack" interfaceversion="1" />
  <privilege partner="CCM" customer="Local" contracttypename="CCMInteractive"
 contracttypeversion="1" interfacename="ComposeDocumentPackGetSuspendedSession"
 interfaceversion="1" />
  <privilege partner="CCM" customer="Local" contracttypename="CCMInteractive"
 contracttypeversion="1" interfacename="ComposeDocumentPackResumeSuspendedSession"
 interfaceversion="1" />
 </application>
</privilegeXML>
```

Set privileges

The `/SetPrivileges` command sets a number of privileges at the same time using an XML file with the privileges that can be obtained with the `/GetPrivileges` command.

The command has two parameters:

- `/PrivilegeXML=` ***Required***. The location of an XML file containing the privileges to load.
- `/Mode=` ***Optional***. Either `replace` or `append`. `replace` removes existing privileges before setting the ones specified in the XML file. `append` adds the privileges to the existing set. When omitted, `append` is used.

The following example demonstrates how to replace the privileges for the application `app` listed in the privileges XML file that resides in `C:\Temp\privileges.xml`.

```
ManageCM /SetPrivileges … /PrivilegeXML=C:\Temp\privileges.xml /mode=replace
```

Remove a privilege for the application

The `/RemovePrivilegeForApplication` removes privileges for the application.

The command has the following parameters:

- `/Application=` ***Required***. The name for the application.

- `/Partner=`. *Required*. The partner for which this application is no longer privileged to call a Contract Manager interface.
- `/Customer=`. *Optional*. The customer for which this application is no longer privileged to call a Contract Manager interface.
- `/ContracTtypeName=`. *Optional*. The contract type name for which this application is no longer privileged to call a Contract Manager interface.
- `/ContractTypeVersion=`. *Optional*. The contract type version for which this application is no longer privileged to call a Contract Manager interface.
- `/InterfaceName=`. *Optional*. The interface name for which this application is no longer privileged to call a Contract Manager interface.
- `/InterfaceVersion=`. *Optional*. The interface version for which this application is no longer privileged to call a Contract Manager interface.

The following example demonstrates how to remove a privilege for the application `app` when the partner is `CCM`, the customer is `local`, and the contract type is `CCMInteractive` version `2`.

```
ManageCM /RemovePrivilegeForApplication … /Application=app /Partner=CCM /Customer=local
 /ContractTypeName=CCMInteractive /ContractTypeVersion=2
```

Add a new contract type to a contract

The `/AddContractTypeToContract` command adds a new contract type to an existing contract.

Note If you add new contract types to the standard contracts, these additions are lost when you upgrade. Standard contracts are those contracts with a partner that can be either KCM or CCM.

This command has the following parameters and flags:

- `/Partner=` *Required*. The partner of the contract.
- `/Customer=` *Required*. The customer of the contract.
- `/Name=` *Required*. Name of the contract type.
- `/Version=` *Required*. Version number of the contract type.
- `/Force` *Optional*.

If an instance is being associated with a contract, the active instance, the instance to be associated, and the previous instance must also support the contract type for this operation to succeed.

Use the `/Force` flag to ignore the previous instance when adding a contract type. If the instance is subsequently rolled back with the command `/RollBackInstance`, and the new functionality from the contract type is used, using the `/Force` flag leads to the functionality breakdown.

The following example demonstrates how the contract type `CCMDistribution` of the version `1` can be added to the contract between the partner `Northwind` and the customer `Futterkiste`.

```
ManageCM /AddContractTypeToContract … /Partner=Northwind
             /Customer=Futterkiste /Name=CCMDistribution /Version=1
```

Remove a contract type from a contract

The `/RemoveContractTypeFromContract` command removes a contract type from a contract.

This command has the following parameters and flags:

- `/Partner=` *Required*. The partner of the contract.
- `/Customer=` *Required*. The customer of the contract.
- `/Name=` *Required*. Name of the contract type.
- `/Version=` *Required*. Version number of the contract type.
- `/Force` *Optional*.

Note If a contract has an active instance associated with it, removing a contract type may remove functionality in use. Use the `/Force` flag to ignore the active instance and implicitly remove the contract type. Also, you cannot remove the last contract type from a contract.

The following example demonstrates how to remove the contract type `CCMDistribution` of the version `1` from the contract between the partner `Northwind` and the customer `Futterkiste`.

```
ManageCM /RemoveContractTypeFromContract …
           /Partner=Northwind /Customer=Futterkiste
           /Name=CCMDistribution /Version=1
```

Register or update one or multiple instances

You can use the `/RegisterInstance` command to register one or multiple instances or update the available interfaces on previously registered instances.

Newly registered instances are marked as available to be associated with contracts. A contract remains associated to an instance when this instance is updated.

This command has the following parameters and flags:

- `/Host=` *Required*. The server hosting the instance.
- `/Instance=` *Required*. The number of the instance or a range to register multiple instances.
- `/Update` *Optional*. Updates instances that are associated with the contract.
- `/Force` *Optional*. Permits breaking updates.

Note The value of the `/Host` parameter should be resolvable both on the Contract Manager server and the Instance server. If the value cannot be resolved, an error message appears.

The number of an instance must be in the range of 1 to 99. A range is separated with a dash mark. For example, `instance=6-99` registers instances from 6 to 99.

If a range of instances is specified, the included instances are registered consecutively. If the registration of an instance fails, the processing stops, and further instances (those of higher numbers) are not processed. A failed instance is not registered or updated. Instances of lower numbers that were already processed successfully remain registered.

You can only update instances associated with a contract using the `/Update` flag.

Also, in an update, you can only introduce new contract types. If an update removes functionality, the command fails. Use the `/Force` flag to force a breaking update.

The following example demonstrates two scenarios: how the instances from `1` to `5` can be registered on the server `host42` or how the registration of already registered instances can be updated. Instances that are associated with a contract are also updated.

```
ManageCM /RegisterInstance … /Host=host42 /Instance=1-5 /Update
```

Show instances

You can use the `/ListInstance` command to get a list of registered instances or to show properties of a particular instance.

This command has the following parameters and flags:

- `/Host=` *Optional*. The server hosting the instance.
- `/Instance=` *Optional*. Lists only this instance.
- `/Available` *Optional*. Lists only instances that are not yet associated with a contract.

If neither `/Host=` nor `/Instance=` is specified, you get a list of all instances for all hosts registered with this Contract Manager. If only `/Host=` is specified, all instances registered for this Contract Manager are listed for this particular host.

Use the `/Available` flag with the parameters `/Host` and `/Instance` to get a list of only those instances that have not yet been associated with any contract. If both `/Host=` and `/Instance=` are specified, the following information is shown:

- Properties
- Availability
- Supported contract types
- A contract that this instance is associated with

Note You cannot specify the parameter `/Instance=` without the parameter `/Host=`.

The following example demonstrates how you can get a list of all registered instances on the server `host42` that have not yet been assigned to any contract.

```
ManageCM /ListInstance … /Host=host42 /Available
```

The second example shows how you can get information on the configuration of the instance `5` on the server `host42`.

```
ManageCM /ListInstance … /Host=host42/Instance=5
```

Change properties of an instance

You can use the `/ChangeInstance` command to change the properties of an instance.

This command has the following parameters and flags:

- `/Host=` *Required*. The server hosting the instance.
- `/Instance=` *Required*. The number of the instance or a range to register multiple instances.
- `/Site=` *Optional*. Changes the KCM ComposerUI for J2EE application name.
- `/Page=` *Optional*. Changes the start page in the specified application.

- `/Force` *Optional*. Permits changes to an instance assigned to the contract.

At least one of the optional parameters must be present. If parameters are omitted, the value of the instance properties is not changed.

You cannot change the properties of an instance active for a contract as this changes the configuration for active users in the production environment. Use the `/Force` flag to ignore the state of the instance.

The following example demonstrates how a start page for the instance `5` on the server `host42` can be changed to `redirect.htm`.

```
ManageCM /ChangeInstance … /Host=host42 /Instance=5 /Page=redirect.htm
```

Remove the registration of an instance

The `/RemoveInstance` command can be used to remove the registration of one or multiple instances.

This command has the following parameters and flags:

- `/Host=` *Required*. The server hosting the instance.
- `/Instance=` *Required*. The number of the instance or a range to remove multiple Instances.
- `/Force` *Optional*. Allows to remove instances that are associated with a contract.

The `/RemoveInstance` command has no impact on the instance, but it can deregister the instance to make it available again for association with a contract.

You cannot remove instances if they are associated with a contract. Use the `/Force` flag to remove an instance that is not yet activated or an instance that was previously associated.

Note You cannot remove the instance that is currently active on a contract.

If a series of instances is specified, the included instances are removed consecutively. If an instance cannot be removed, the processing stops, and further instances (those of higher numbers) are not processed. The failed instance is not removed. Instances of lower numbers that have been already processed successfully are removed.

The following example demonstrates how the instances in the range from `1` to `5` can be removed from the server `host42`.

```
ManageCM /RemoveInstance … /Host=host42 /Instance=1-5
```

Associate an instance with a contract

The `/AssociateInstance` command associates an available instance with a contract.

This command has the following parameters and flags:

- `/Partner=` *Required*. The partner of the contract.
- `/Customer=` *Required*. The customer of the contract.
- `/Host=` *Required*. The server hosting the instance.
- `/Instance=` *Required*. The number of the instance.
- `/Activate` *Optional*. Activates the instance after it is associated with a contract.

- `/OnLine` *Optional*. On activation, configures the instance to be used with KCM ComposerUI for J2EE instead of KCM ComposerUI for HTML5.
- `/Force` *Optional*. Discards a non-activated instance already associated with the contract.

The association makes the instance unavailable for association with other contracts. To make an instance available again, you need to deregister it and register it again.

If another instance associated with the contract is not activated yet, the command fails. Use the `/Force` parameter to abandon the previous association.

The associated instance does not start processing for the contract until you activate it with `/ActivateInstance`. You can use the `/Activate` flag immediately after the association is completed successfully.

The following example demonstrates how the contract between the partner `Northwind` and the customer `Futterkiste` can be unconditionally associated with the instance `4` on the server `host42`, and how processing of the contract can be switched to this instance.

```
ManageCM /AssociateInstance … /Partner=Northwind /Customer=Futterkiste
 /Host=host42 /Instance=4 /Force /Activate
```

Remove instance associations from a contract

The `/RemoveInstanceAssociation` command removes all instance associations from a contract.

This command has the following parameters and flags:

- `/Partner=` *Required*. The partner of the contract.
- `/Customer=` *Required*. The customer of the contract.
- `/Force` *Optional*. Disassociates active instances.

If there is an active instance for the contract, the `/Force` flag is required to remove the associations.

The following example demonstrates how the instance associations belonging to the partner `Northwind` and the customer `Futterkiste` can be unconditionally removed.

```
ManageCM /RemoveInstanceAssociation … /Partner=Northwind /Customer=Futterkiste /Force
```

The `/RemoveInstanceAssociation` command does **not** make an instance available for another contract. To make the instance available, first deregister it with the `/RemoveInstance` command and then register it again with the `/RegisterInstance` command. For more information, see Remove the registration of an instance and Register or update one or multiple instances, respectively.

Switch processing of a contract to the latest associated instance

The `/ActivateInstance` command switches processing from the current instance, if any, to the instance that was last associated with the contract using the `/AssociateInstance` command.

The `/ActivateInstance` command does not require a reload of the Contract Manager configuration.

This command has the following parameters and flags:

`/Partner=` *Required*. The partner of the contract.

`/Customer=` *Required*. The customer of the contract.

`/Host=` *Required*. The server hosting the instance.

`/Instance=` *Required*. The number of the instance.

`/OnLine` *Optional*. Configures the instance to be used with KCM ComposerUI for J2EE instead of KCM ComposerUI for HTML5.

This command fails if the `/Host=` and `/Instance=` parameters do not match the instance to switch to.

If an instance that already performs the processing for this contract (the current instance), a reference to this instance is kept. Use the `/RollBackInstance` command to switch processing back to this instance.

The following example demonstrates how processing for the contract between the partner `Northwind` and the customer `Futterkiste` can be switched to the instance `4` on the server `host42`.

```
ManageCM /ActivateInstance … /Partner=Northwind /Customer=Futterkiste
         /Host=host42 /Instance=4
```

Switch processing of a contract to the previous instance

You can use the `/RollBackInstance` command to switch processing for a contract back to the previous instance.

The `/RollBackInstance` command does not require a reload of the Contract Manager configuration.

This command has the following parameters and flags:

`/Partner=` *Required*. The partner of the contract.

`/Customer=` *Required*. The customer of the contract.

`/Force` *Optional*. Discards a non-activated instance already associated with a contract.

You can only roll back a contract if there is a previous instance to switch to. You cannot roll back twice without activating another instance.

The instance active for this contract when the `/RollBackInstance` command is executed will be reset to the status "pending activation." The instance that was marked as previous for this contract is activated. To reactivate the instance that was rolled back, you can use the `/ActivateInstance` command.

If another instance is associated with the contract awaiting activation that is neither the active instance nor the previous instance, the `/RollbackInstance` command fails. Use the `/Force` flag to activate the previous instance, set the active instance to "pending activation," and abandon the association that was not yet activated.

The following example demonstrates how processing for the contract between the partner `Northwind` and the customer `Futterkiste` can be switched back to the previous instance.

```
ManageCM /RollBackInstance … /Partner=Northwind /Customer=Futterkiste
```

Export the Contract Manager internal database to a file

You can use the `/ExportDatabase` command to write the Contract Manager internal database or a part of the internal database to a file for transfer or maintenance purposes. This file can be imported with the `/ImportDatabase` command.

This command has the following parameters and flags:

- `/File=` *Required*. The output file.
- `/CMVersion=` *Optional*. Specifies a different version of the Contract Manager. You can use this parameter to export Contract Manager internal databases from older Kofax Communications Manager versions.
- `/State` *Optional*. When exporting contracts, it exports information about the instances associated with the contracts.

By default, all customization in the Contract Manager internal database is exported. You can specify a filter to limit the amount of data to be exported. By default, all standard contract types are filtered out. Standard contract types start with "CCM" or "KTA" and are automatically added during a standard installation.

The following filters are available:

- Contracts

  `/Contracts` *Required*. Selects this filter.

  `/Partner=` *Optional*. Filters the partners.

  `/Customer=` *Optional*. Filters the customers of a partner. Requires `/Partner=`. This export includes all contract types and interfaces used in the exported contracts.
- Contract Types

  `/ContractTypes` *Required*. Selects this filter.

  `/Name=` *Optional*. Filters the name of the contract type.

  `/Version=` *Optional*. Filters the versions of a contract type. Requires `/Name=`. This export includes all interfaces used in the exported contract types.
- Interfaces

  `/Interfaces` *Required*. Selects this filter.

  `/Name=` *Optional*. Filters the name of the interface.

  `/Version=` *Optional*. Filters the versions of an interface. Requires `/Name=`.
- Instances

  `/Instances` *Required*. Selects this filter.

  `/Host=` *Optional*. Filters the instances on specific hosts.
- Applications

  `/Applications` *Optional*. If the `/Applications` flag is passed, only applications are exported.

Note You can specify only one filter for an export.

The `/Partner=`, `/Customer=`, `/Name=`, and `/Host=` flags use the Transact-SQL LIKE statement to provide limited wildcard searches. Use the `%` sign to specify a match for any characters or the `_` character to match a single character.

Instance information is only exported when contracts are exported with the `/State` flag or when instances are explicitly filtered with the `/Instances` flag. When instance information is included in the export, you can retrieve this information with the `/ImportDatabase` command.

The files exported with the `/ExportDatabase` command are intended to be imported with the `/ImportDatabase` command.

Note Manual creation or modification of the XML files is not supported.

The following example demonstrates how to export the full Contract Manager database to the file `Northwind.xml`, including all registered instances and their associations with contracts.

```
ManageCM /ExportDatabase … /File=Northwind.xml
         /State
```

The second example shows how to export the contract between the partner `Northwind` and the customer `Futterkiste`.

```
ManageCM /ExportDatabase … /File=Export.xml
         /Contracts /Partner=Northwind /Customer=Futterkiste
```

The third example presents how to export contracts belonging to the partner `Northwind` with names begin with `Futt`.

```
ManageCM /ExportDatabase …
         /File=Export.xml
         /Contracts /Partner=Northwind /Customer=Futt%
```

The fourth example demonstrates how to export all contract types with names begin with `ISV`.

```
ManageCM /ExportDatabase …
         /File=Export.xml
         /ContractTypes /Name=ISV%
```

The fifth example shows how to export all instances from `host42` registered with the KCM `5.1` Contract Manager.

```
ManageCM /ExportDatabase … /File=Export.xml
         /CMVersion=5.1
         /Instances /Host=host42
```

Import content from a file to the Contract Manager internal database

The `/ImportDatabase` command imports objects to the Contract Manager internal database that were previously exported using the `/ExportDatabase` command.

This command has the following parameters and flags:

- `/File=` *Required*. The input file.
- `/Update` *Optional*. Updates existing objects.
- `/Offline` *Optional*. Ignores any instances that ManageCM cannot connect to.

The `/ImportDatabase` command imports all objects from the specified file. If the file includes instances, they are registered or updated to reflect the availability of new interfaces and contract types.

If one of the objects could not be imported, the import is cancelled, and the Contract Manager internal database is not modified.

Use the `/Update` flag if one or more of the objects are already present in the Contract Manager internal database. Updates are allowed only when the import extends the existing object.

Use the `/Offline` flag to ignore any instances that the import process fails to connect to. If such an instance is associated with the contract, the import fails. This flag supports transfer of configurations to offline servers hosting retired instances.

The files imported with `/ExportDatabase` are expected to have been exported with the `/ExportDatabase` command.

Note Manual creation or modification of the XML files is not supported.

The following example demonstrates how all objects from the `Export.xml` file can be imported.

```
ManageCM /ImportDatabase …
         /File=Export.xml
```

Find or change the location of the shared resources folder

You can use the `/SetSharedResource` command to determine or change the location where the Contract Manager stores shared objects for import and export.

This example shows how to retrieve the current location of the folder.

```
ManageCM /SetSharedResource
```

To change the current location, specify the following parameter:

- `/folder=` *Optional*. The new location for the shared resources folder.

  Note The change only takes effect after a restart of the Apache Tomcat CCMRuntime instance.

The following example demonstrates how to change the folder location to `g:\storage\shared`.

```
ManageCM /SetSharedResource /folder=g:\storage\shared
```

Specify interfaces

An interface in the Contract Manager corresponds to a web service and exposes a service in KCM Core. In some cases, an interface corresponds to a different KCM component. The interface is identified by a unique combination of the name and the version and specifies a number of input and/or output parameters, together with a set of attributes that influence its behavior.

Interfaces are created using an XML file. The interface.xsd file, which resides in `<deploy root>\KCM\Documentation\<KCM version>\Resources\Schemas\ManageCM`, defines the exact format for this file.

An interface has the following attributes:

- Name. Must contain only alphanumeric characters and periods.
- Version.
- Type. Use one of the following values:

  `Core`. This interface only interacts with KCM Core.

  `ComposerUIHtml5`. This interface is to interact with ComposerUI for HTML5, alongside its service implementation in KCM Core.

  Note If your interface has the type `ComposerUIHtml5`, you do not need to specify output parameters for `ccmsessionid` and `url` as they are implicitly defined for this interface type. You do need to return `sessionID` in the interface script that implements the interface.

  `ComposerUIJ2EE`. This interface is to interact with KCM ComposerUI for J2EE, alongside its service implementation in KCM Core.

  `Designer`. This interface is to interact with KCM Designer.

  `Administration`. Indicates that this interface is to perform administrative actions.
- Session specifier. Use one of the following values:

  `Require`. This interface requires a session in the Contract Manager to be present.

  `Create`. This interface creates a session in the Contract Manager.

  `None`. This interface does not interact with or does not require any session in the Contract Manager.
- UseLegacySessionId flag. *Optional*. By default, false. If this flag is set and the interface performs actions with a session in the Contract Manager, the session ID is identified with its old name *itpcloudid* instead of *ccmsessionid*.
- CoreScript. Specifies which KCM Core service must be called when the web service corresponding to this interface is called. This element is required for Core, ComposerUIHtml5, and ComposerUIJ2EE interfaces only.
- Parameters that contain two elements: input and output. Both contain a list of `Param` elements, specifying the following for each parameter of the interface:

  Name. Must contain only alphanumeric characters and periods.

  Type, indicating the type of the parameter. Must be any of the following: `Boolean`, `Document`, `InteractiveResult`, `LtbURI`, `Message`, `Name`, `RepositoryObjectName`, `URL`.
- Optional. Indicates whether the parameter is optional.

Note The order of the input parameters for the interface must correspond to the order of the KCM Core script parameters.

An exception to the ordering rule is parameters of type `Document`. They are not script parameters but are implemented as calls to the `ReceiveFile` function in KCM Core scripts. Therefore, these parameters are skipped when passing the interface parameters on to KCM Core. Another exception is the `RedirectUrl` parameter that can be added to KCM Core scripts for KCM ComposerUI interfaces.

Interfaces can update an existing interface when it meets the following requirements:

- All existing parameters are still present in the same order.
- Existing parameters can be redefined as optional, but only when they are followed by required parameters in their ordering.
- New parameters are added to the end of the list and are optional.

Specify contract types

A contract type in the Contract Manager corresponds to a collection of interfaces. A contract type is identified by a unique combination of the name and the version. It is used to generate the WSDLs for the interfaces.

Contract types are created using an XML file. The ContractTypeV1.xsd file, which resides in `<deploy root>\KCM\Documentation\<KCM version>\Resources\Schemas\ManageCM`, defines the exact format for this file.

A contract type has the following attributes:

- Name. Must be a valid XML element name, not exceeding 50 characters. Can only contain alphanumeric characters.
- Version.
- Type. With one of the following values:

  `Core`. Indicates that this contract type only contains Core, Designer, or Administration interfaces.

  `ComposerUIHtml5`. Indicates that this contract type contains interfaces only for Core, Designer, Administration, or ComposerUI for HTML5.

  `ComposerUIJ2EE`. Indicates that this contract type contains interfaces only for Core, Designer, Administration, or KCM ComposerUI for J2EE.
- `Interfaces`. Contains one or more interface elements, each containing the interface name and the interface version.

Before a contract type is created, it is verified that each referenced interface exists and that the interface type is compatible with the type of the contract type. A contract type can update an existing contract type only if all existing interfaces are still present.

Note When an interface is created or updated, you must reload the Contract Manager configuration using the `/ReloadConfiguration` command (see Reload the configuration).

Chapter 4

Use the ConfigureInstance tool

This chapter gives information about the ConfigureInstance tool functionality.

The ConfigureInstance tool is designed to transfer interface implementations and configurations between KCM instances and perform maintenance tasks on the instances. You can use the tool to export the configuration from an instance, deploy the configuration to one or multiple new instances, update the interface implementations on another instance, and replace licenses.

The export of the configuration includes custom interfaces, scripts, and compiled implementations. Also, you can include external files to facilitate interfaces deployment.

The ConfigureInstance tool resides in:

`<deploy root>\KCM\Programs\<KCM version>\ITP Server\bin`

The `ConfigureInstance` command line has the following structure.

```
ConfigureInstance <cmd> <par>
```

`cmd`: A flag that defines a command to be performed. A flag is used to enable features.

It is written as `/flag`.

`par`: Zero, one, or multiple parameters and flags that apply to the `ConfigureInstance` command. A parameter is written as `/parameter=value` to define a value of the parameter. If the value contains spaces, the parameter should be enclosed in quotes.

Note Flags and parameters are case-insensitive.

An example is provided here.

```
ConfigureInstance /Export /Version=5.4 /Instance=1 /File=export.xml
```

Also, the ConfigureInstance tool provides short help on the command line with the `/?` parameter.

- `ConfigureInstance /?` shows the available commands.
- `ConfigureInstance /command /?` provides help about the `/command` command.

Consistency checks

The `ConfigureInstance` command performs consistency checks prior to performing a requested operation. If the operation fails, the changes to the failed instance are rolled back. Instances that have been already successfully updated are not rolled back.

Note All modified files are backed up to a time-stamped copy in the same location.

The configuration data includes library .cds files that contain pre-compiled interface implementations. Some of these library files can be provided by ISVs, implementing custom interfaces.

All the library files are tagged with KCM Core versions that the library can be deployed with. If a particular KCM Core version is not supported by the library, the operation is cancelled. You can use the `/Force` flag to force an update, but you have to replace the conflicting library file by a supported version from the ISV prior to starting the instance.

Export custom Data

You can include additional files with the `/Export` command. This can be data per instance inside the instance work directory or global outside the instance directory tree.

All included files are restored during the `/Import` and `/Update` operations.

Commands

This section lists and describes the ConfigureInstance tool commands.

This table lists the ConfigureInstance tool commands, gives a short description, and provides links to the corresponding sections.

Parameter	Usage	Link to the Section
`/Export`	Writes the configuration of an instance to a file for later processing with the `/Import` and `/Update` commands.	Export an instance
`/Import`	Deploys a previously exported configuration to one or multiple instances.	Import an instance
`/Update`	Deploys the interface configuration from a previously exported configuration to another instance.	Update an instance
`/ReplaceLicense`	Replaces the licenses on one or more instances.	Update the licenses

Export an instance

The `/Export` command writes the configuration of an instance to a file for later processing with the `/Import` and `/Update` commands.

This command has the following parameters:

- `/Version=` *Required*. Specifies the KCM version for the instance to be exported.

- `/Instance=` *Required*. Specifies the instance to be exported.
- `/File=` *Required*. The file to which the instance is exported.
- `/PathGlobal=` *Optional*. A comma-separated list of directories. All content in these directories is included in the export. These directories should be absolute paths.
- `/PathInstance=` *Optional*. A comma-separated list of directories. All content in these directories is included in the export. These directories should be relative to the instance work directory.

The `/Export` command fails if one of the directories in the `/PathGlobal=` and `/PathInstance=` settings does not exist.

Note The data written with the `/Export` command is intended to be processed with the other commands. The content is not editable.

The following example demonstrates how the instance `1` from the KCM version `5.3` can be exported to the `d:\ExportInstance.xml` file.

```
ConfigureInstance /Export /Version=5.3 /Instance=1
   /File=d:\ExportInstance.xml
   /PathGlobal=c:\KCM\Tools,c:\Support
   /PathInstance=Did
```

This export includes all content of the following directories:

```
C:\KCM\Tools
```

```
C:\Support
```

```
C:\KCM\Work\5.3\Instance_01\core\Did
```

Import an instance

The `/Import` command deploys a previously exported configuration to one or multiple instances. This operation overwrites all settings on the instance, including the environment settings and the connection configuration.

The command is intended to deploy a pre-configured configuration to one or multiple new instances. The same configuration is deployed to all instances, so you might need to manually edit the settings to apply a configuration that differs for each instance.

This command has the following parameters:

- `/Version=` *Required*. Specifies the KCM version the instances of which are to be updated.
- `/File=` *Required*. The file from which the instance is read. This file must be created with the `/Export` command.
- `/Instance=` *Optional*. The number of an instance or a range to register multiple instances. If this parameter is omitted, all instances are updated.
- `/Force=` *Optional*. Permits an import even when libraries are incompatible with the specified KCM version. This can result in a KCM instance that cannot be used until the libraries are updated with supported versions.

If the export includes external files, they are deployed to the global directories and to each of the instance directories.

The following example shows how the configuration can be deployed from the `d:\ExportInstance.xml` file to the instances `1` to `4` belonging to KCM `5.4.0`.

```
ConfigureInstance /Import /Version=5.4 /Instance=1-4
   /File=d:\ExportInstance.xml
```

Update an instance

The `/Update` command deploys the interface configuration from a previously exported configuration to another instance. This operation does not overwrite other settings on the instance, including the environment settings and the connection configuration.

The intended use of this command is to migrate interface definitions between two configured instances.

This command has the following parameters:

- `/Version=` *Required*. Specifies the KCM version for the instances to be updated.
- `/Instance=` *Required*. The number of the instance to be updated.
- `/File=` *Required*. The file from which the instance is read. This file must be created using the `/Export` command.
- `/Force=` *Optional*. Permits an update when libraries are incompatible with the specified KCM version. This can result in a KCM instance that cannot be used until the libraries are updated with supported versions.

If the export includes external files, existing files are left unmodified. Only the files that do not yet exist on the target instance are deployed. KCM Core service constants are not changed on the target instance. Constants are added to the export file if they do not yet exist.

The following example demonstrates how the interfaces can be exported from `d:\ExportFromDev.xml` to the instance `1` belonging to KCM `5.4.0`.

```
ConfigureInstance /Update /Version=5.4 /Instance=1
   /File=d:\ExportFromDev.xml
```

Update the licenses

The `/ReplaceLicense` command updates the licenses for one or more instances on a server.

This command has the following parameters:

- `/Version=` *Required*. Specifies the KCM version of the instance to be updated.
- `/File=` *Required*. Specifies the XML license file that must be deployed.
- `/Instance=` *Optional*. The number of an instance or a range to update multiple instances. If this parameter is omitted, all instances are updated.
- `/Manual` *Optional*. Updates the licenses but does not restart running services.
- `/Start` *Optional*. Updates the licenses and restarts all services.

The `/ReplaceLicense` command restarts all running services and ignores services that are stopped or disabled. The `/Manual` and `/Start` flags override this behavior.

The files containing license data are backed up before the services are restarted. If a service fails to restart, the files are restored to their old state, and you might need to manually restart the services. When

updating multiple instances with `/ReplaceLicense`, the command does not roll back services that were successfully updated prior to a failure.

The following example shows how to update all licenses on a KCM `5.4.0` server with the license data in the file `c:\UpdatedLicense.xml`.

```
ConfigureInstance /ReplaceLicense /Version=5.4
   /File=c:\UpdatedLicense.xml
```

Chapter 5

Manage language packs

KCM supports eight languages: English, German, French, Italian, Spanish, Japanese, Brazilian Portuguese, and Dutch. For KCM ComposerUI for HTML5, the set of supported languages is extensible. To add a language to ComposerUI for HTML5, you can enhance a KCM installation with a language pack.

In KCM, you can retrieve, translate, add, list, and remove language packs. To manage language packs, use the following scripts that reside in `<deploy root>\KCM\Programs\<KCM version>\Management`:

- GetLanguagePack.ps1
- AddLanguagePack.ps1
- ListLanguagePacks.ps1
- RemoveLanguagePack.ps1

Note If upgrading your KCM installation, you need to reinstall any custom additional language packs. We recommend that you execute the script GetLanguagePack.ps1 prior to upgrading the product, update the custom language packs so they include new translations, and then install the language packs using the script AddLanguagePack.ps1.

Retrieve a language pack

To get a language pack for a particular language, start the Windows PowerShell command prompt and execute the script GetLanguagePack.ps1.

It has two parameters:

- `Destination!Path` The path to the destination directory in which the results are placed. The script automatically creates this directory.
  The output directory structure contains files, such as <language code>.js and <language code>_dlg.js.
- `Language!Code` The language code of the target language. If omitted, language code "en" is used.

Note We recommend that you use the "en" language as a base for your translations as the other localizations may lag behind.

Example

```
GetLanguagePack.ps1 Destination!Path=C:\English Language!Code=en
```

Translate a language pack

A language pack consists of a directory structure containing .js files, which include translations for a specific language.

Note The contents of the language files should be valid JavaScript and properly escaped according to the JavaScript rules. Also, if translations contain special characters, they may not be shown properly on the user interface in all cases. In such a case, verify that the corresponding file in the language pack contains a byte order mark.

1. To retrieve a language pack, execute the script GetLanguagePack.ps1. For information on the script parameters, see Retrieve a language pack.
2. In the directory ComposerUI, rename the .js file, replacing the source language code with the language code of the target language.

   Example For Finnish, rename en.js and en_dlg.js to fi.js and fi_dlg.js, respectively.
3. In the directory ComposerUI, edit the .js files as follows.

   Note The language code must be lowercase.

   - Replace all occurrences of ["source language code"] with ["target language code"].

     Example For Finnish, replace ["en"] with ["fi"].
   - Below the section ITPiLocalization.translations, translate the values of all key/value pairs (formatted as "key": "value") in the target language.
   - Below the section ITPiLocalization.datepicker, translate the values of all key/value pairs (formatted as "key": "value") in the target language.
   - If applicable, below the section ITPiLocalization.formats, modify the formatting strings to reflect correct date, time, and number for the target language.
4. In the TinyMCEPlugins directory structure, rename each file, replacing the source language code with a language code of the target language.
5. In the TinyMCEPlugins directory structure, edit each file in the following way:
   - Replace the source language code on the first line with the target language code.
   - Each file consists of key/value pairs (formatted as: "key": "value"). Edit the values with the target language translations.
6. Download the language pack for the target language from the TinyMCE website:

   http://archive.tinymce.com/i18n3x/index.php?ctrl=lang

   Overwrite the files in the TinyMCEStandard directory structure with the files you downloaded.
   - If the target language is not available from the TineMCE website or custom localizations are required, follow the same procedure as in Steps 4 and 5 for the TinyMCEStandard directory.

Add a language pack

To add your own language pack for a particular language, start the Windows PowerShell command prompt and execute the script AddLanguagePack.ps1.

The script has two parameters:

- `Source!Path` The path to the source directory that contains the translation of an existing language pack.
- `Language!Code` The language code of the target language.

You cannot add or overwrite standard language codes, which include 'de', 'en', 'es', 'fr', 'it', 'ja', 'nl', 'pt-br'.

Example

```
AddLanguagePack.ps1 Source!Path=C:\Finnish Language!Code=fi
```

Note If an error occurs when you are overwriting an existing language pack, it means that this language pack file is currently in use. If such a situation occurs, try to overwrite the language pack later. Alternatively, you can stop the service "Apache Tomcat CCMRuntime instance <version number>" to interrupt all current sessions. Then retry to overwrite the language pack and restart the service.

Note If any message for the target language is missing from the respective custom language pack, TinyMCE shows the message ID.

List language packs

To list the language codes for the language packs currently available for KCM ComposerUI for HTML5, start the Windows PowerShell command prompt and execute the script ListLanguagePacks.ps1.

Language codes belonging to the standard languages are marked with (standard language).

It has one optional parameter:

- `Destination!File` If a destination file is given, the list is also written to this destination file. The path in which the destination file is written should exist. This file itself should not yet exist.

Example

```
ListLanguagePacks.ps1 Destination!File=c:\temp\languagelist.txt
ListLanguagePacks.ps1
```

Example output:

```
Language Packs Language Codes
-----------------------------------------
de (standard language)
en (standard language)
es (standard language)
fr (standard language)
it (standard language)
ja (standard language)
nl (standard language)
```

```
pl
pt-br (standard language)
pt
```

Validate a language pack

You can test the generated translations. To do so, pass the code of an applied language pack on the "locale" run option on the Start function of the ComposerUI JavaScript API. The translations in the language pack are then used during the document/Document Pack composition.

When testing the translations, consider the following:

- Forms and QForms containing questions of different types (text, (textblock) choice, checkbox, number, checkbox, date, time)
- Question validation on Forms
- Content Wizards containing different types of selections (section/textblock, single/multiple)
- Letter Books
- Document Pack Templates
- Editable Text Blocks. They are presented to the user in a TinyMCE-based editor. The TinyMCEPlugins folder in a language pack contains KCM-specific translations in the context of this editor. Most of the translations are visible as tooltips on the menu or in the dialog boxes that appear when the user clicks a menu item.

Note The "itpmetadata" plugin is currently not used by ComposerUI.

Remove a language pack

To remove a non-standard language pack, start the Windows PowerShell command prompt and execute the script RemoveLanguagePack.ps1.

The script has one parameter:

- `Language!Code` The language code of the target language.

Example

```
RemoveLanguagePack.ps1 Language!Code=fi
```

Note If an error occurs when you are removing an existing language pack, this could mean that this language pack is currently in use. If such a situation occurs, try to remove the language pack at a later time. Alternatively, you can stop the service "Apache Tomcat CCMRuntime instance <version number>" to interrupt all current sessions. Then, retry to remove the language pack and restart the service.

Chapter 6

Configure KCM settings on Docker

When a KCM instance is installed on Docker, the KCM Core Administrator is not available. Instead, a number of command line configuration tools are available to configure KCM settings. This chapter provides information on the following tools:

- Use the CompileInstanceScripts tool
- Use the ConfigureInstanceSettings tool
- Use the ManageInstanceEnvironment tool
- Use the RestartServices tool
- Manage language packs on Docker
- Other management tools on Docker

Use the CompileInstanceScripts tool

In the Docker environment, to update and compile the constants, user scripts, and user library scripts for KCM Core, use the CompileInstanceScripts tool. This tool does not operate on a KCM Core instance directly. Instead, it uses a folder on the Docker container. This allows the synchronization between two locations using a pull operation and push operation. Between these operations, the folder on the Docker container can be manipulated.

Note The scripts folder must follow certain rules to be compiled and applied successfully. We recommend that you always start with a pull command to get this folder as it has the correct structure and contains the required exit points. In addition, this folder shows the KCM Core services currently running on the instance, so that you can avoid pushing an incomplete set of scripts.

Pull scripts

You can use the CompileInstanceScripts tool to pull the complete set of the KCM Core scripts and constants of the running instance container into the given folder. It will populate the folder with the following components:

- The script specification file. This XML file contains the constant definitions as well as the KCM Core service definitions.
- The XSD file that the script specification file needs to adhere to.
- The scripts that implement the KCM Core services.

For more details about the KCM Core scripting, see the *Kofax Communications Manager Core Scripting Language Developer's Guide*.

Push scripts

You can use the CompileInstanceScripts tool to push the scripts and the specification file from the given folder into the running Docker instance container, which will compile the scripts.

KCM Core will not use the newly pushed scripts until they are applied successfully. When pushing these scripts, set the `Script!Apply` parameter to true. The scripts will be applied if no compilation errors occur.

For more details, see the *Kofax Communications Manager Core Scripting Language Developer's Guide*.

The following table lists and describes the CompileInstanceScripts tool commands.

Parameter	Description
`scripts!Direction=<pull or push>`	Required. Defaults to push if no `scripts!Direction` is given. • **Pull**: Stores the scripts as active on the container in the given `Scripts!Folder`. Writes the specifications to the CompileInstanceScripts.xml file containing the script specification, as well as to the CompileInstanceScripts.xsd file describing the structure of the XML document that the script specification file needs to adhere to. Any changes in this folder that are not yet successfully pushed and applied will be overwritten by the currently active scripts. • **Push**: Pushes the scripts from the given `Scripts!Folder` and the CompileInstanceScripts.xml specification file to the Docker container and compiles the scripts according to the specifications. These scripts are applied if the compilation succeeds and the parameter `Scripts!Apply` is set to true.
`Scripts!Folder`	Required. The folder containing the scripts to update. When pushing, the folder must contain the following components: • The compileinstancescripts.xml file that adheres to the compileinstancescripts.xsd file. • The Userlibrary subfolder with the Userlibrary scripts.
`Scripts!Apply=<true or false>`	Optional. By default, false. When set to true, applies the results of the script compilation to KCM Core. Only applicable when `Scripts!Direction` is set to `push`. The result is applied only if no compilation errors are reported.
`Instance!Number`	Optional. Indicates the instance for which scripts should be compiled. Default is 1 if no `Instance!Number` parameter is given. On Docker, this parameter must be omitted. It is provided for future compatibility with non-Docker installations.

See the following example of the specification file:

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration Version="5.0" xmlns="urn:KCCMConfigExport5">
 <Instance Version="5.4" >
```

```
  <Environment key="WatermarkPdfConcept" value="watermarkconcept.pdf" />
  <Environment key="AllowRepositoryModelRun" value="Y" />
  <Environment key="_ValidateInput" value="N" />
  ...
  <Environment key="MyConstant" value="myvalue>
  <Service Name="RunMdl">
   <Parameter Name="Model" Value="$1" />
   <Parameter Name="Result" Value="$2" />
   <Parameter Name="Keys" Value="$3" />
   <Parameter Name="Extras" Value="$4" />
   <Parameter Name="Environment" Value="$5" />
   <Parameter Name="MetaData" Value="$6" />
   <Parameter Name="DBB_XMLInput" Value="$7" />
   <Parameter Name="DBB_XMLOutput" Value="$8" />
  </Service>
  <Service Name="RunDocumentPackTemplate">
   <Parameter Name="DocumentPackTemplate" Value="$1" />
   <Parameter Name="Result" Value="$2" />
   <Parameter Name="Environment" Value="$3" />
   <Parameter Name="DBB_XMLInput" Value="$4" />
  </Service>
  ...
  <Service Name="MyScripts">
   <Parameter Name="MyParameter" Value="$1" />
  </Service>
 </Instance>
</Configuration>
```

Add constants

To add new constants to the specification file, add the following line to the list with environment nodes:

`<Environment key="<name of the constant>" value="<value of the constant>" />`.

For `<name of the constant>`, enter the name for the constant to add.

For `<value of the constant>`, enter the value for the constant to add.

Note Do not remove `WatermarkPdfConcept`, `AllowRepositoryModelRun, ValidateInput`, or alter the value, unless you want to alter the behavior of KCM Core.

`AllowRepositoryModelRun` with the value `y` indicates that templates can be tested from KCM Designer. The other constants are used in legacy scripts.

Add scripts

To update the specification file with the KCM Core services, add the following lines to the list with service nodes:

- ```
  <Service Name="<name of the script to add>" >
  </Service>
  ```

  For `<name of the script to add>`, enter the name for your script. A file with the same name and the extension .dss should be present in the `Scripts!Folder`.
- ```
  <Service Name="<name of the service to add>" >
  </Service>
  ```

  For `<name of the service to add>`, enter the name for your service. A script file with the same name and the extension .dss should be present in the `Scripts!Folder`.

If the script includes any parameters, add the following lines:

```
<Service Name="<name of the script to add>" >
<Parameter Name="<name of the parameter to add>" Value="<$#>" />
</Service>
```

- For `<name of the parameter to add>`, enter the name for the parameter in your script.
- For `<$#>`, enter a sequence number, such as $1.

Parameters in a script are known by their name. However, parameters in a job are known by their sequence number. The attributes of the Parameter tag provide a mapping between name and sequence number.

Note Only alphanumerical characters are supported in parameter values. For more information, see the *KCM Core Scripting Language Developer's Guide*.

Use the ConfigureInstanceSettings tool

This section provides information on the following commands:

- Get help for configuration settings type
- Get help for runtime settings type
- Runtime settings levels
- Request runtime settings per level
- Request runtime settings per setting type
- Set values for configuration settings
- Set values for runtime settings
- Remove runtime settings

In the Docker environment, use the ConfigureInstanceSettings tool to configure KCM Core instance settings. Offering similar functionality as KCM Core Administrator on a regular installation, the tool itself also works on a regular installation.

The ConfigureInstanceSettings tool supports a number of different actions, such get help on one or more settings, request a list of settings, set a value for one or more settings, and remove a setting. Only one action can be executed at one call to the ChangeInstanceSettings tool.

If you want to apply your changes to the settings, restart the services by running the RestartServices tool. For more information, see Use the RestartServices tool.

If you start the ConfigureInstanceSettings tool without any parameters, it gives you an overview of all configuration types that it manages. The following types of configuration can be defined:

- DPManager
- CoreLogging
- Monitor
- HttpMonitor
- JobRecovery
- JobScheduling

- Watcher
- ClientConnection
- CoreManager
- RuntimeSettings

The RuntimeSettings type forms a special category of settings. The settings in this category can be used to configure the templates behavior at runtime, which requires working knowledge of template composition. All of the other settings are related to the KCM Core document processor infrastructure where the templates are run.

Note There is a difference between viewing the values for a specific setting type and all settings types:

- The list of values for a specific setting type shows all settings.
- The list of values for all setting types shows only those settings that deviate from the default. Otherwise, the list of all setting types would be very long.

Get help for configuration settings type

You can view more information on the configuration types by entering the following command:

```
ConfigureInstanceSettings Help!Configuration=<Configuration Type>
```

This command gives you an overview of all configuration settings that can be found for a particular configuration type and a short description of their use.

An overview of the values of the configuration settings can be viewed by using the following command:

```
ConfigureInstanceSettings List!Configuration=<Configuration Type>
```

This command lists the values of the settings for the given configuration type.

To see a list with all available configuration settings, enter the following command:

```
ConfigureInstanceSettings List!Configuration=Settings
```

This command returns a list of all settings except the runtime settings.

Get help for runtime settings type

The RuntimeSettings configuration type is a special category that has its own subset of configuration areas. Enter the following command to get an overview of the runtime configuration areas within this category:

```
ConfigureInstanceSettings Help!Configuration=RuntimeSettings
```

In the RuntimeSettings category, the following areas can be found:

- General: General settings
- ODBC: ODBC connection
- OCI: Oracle connection
- XMLFile: XML File connection
- XMLWEB: XML Web connection

- XMLMQ: XML MQSeries connection
- Mainframe: Mainframe connection
- AS400: AS/400 connection
- Local: local connection

To get specific help on each of these areas, enter the following command:

```
ConfigureInstanceSettings Help!Configuration=RuntimeSettings RuntimeSettings!
SettingType=<RuntimeSetting type>
```

This command provides a list of all settings available for that type of setting with their default value and a short description of their use.

Runtime settings levels

The runtime settings can be set for various levels:

- Global level. Settings that are applied globally.
- Environment level. Settings that are only applied for an environment.
- DID level. Settings that are only applied for a DID in an environment.
- DID module level. Settings that are only applied for a DID module in a DID within an environment.

On a global level, multiple environments are available. Each of them can have multiple DIDs. Finally, each DID can have multiple DID modules defined in it. Creating and maintaining this hierarchy can be done by using the ManageInstanceEnvironment tool. For more details, see Use the ManageInstanceEnvironment tool.

Settings inherit values from the other levels according to a specific order, until they get overridden. If a setting is set to a particular value at a particular level, this same value will be used for this setting on all of the other levels within it, based on the following logic:

- If no value is given for a setting at the DID module level, the value of the setting at the DID level is used.
- If no value is given for a setting at the DID level, the value of the setting at the environment level is used.
- If no value is given for a setting at the environment level, the value of the setting at the global level is used.

Each setting has a default value. If you specify no value for a particular setting at any level, the default value is used for that setting.

Set values for different levels:

- Set a value for a particular environment using the following parameter:

  ```
  RuntimeSettings!Environment=<Environment Name>
  ```

- Set a value for a particular DID using the following combination of parameters:

  ```
  RuntimeSettings!Environment=<Environment Name> RuntimeSettings!Did=<Did
  Name>
  ```

- Set a value for a particular DID module using the following combination of parameters:

  ```
  RuntimeSettings!Environment=<Environment Name> RuntimeSettings!Did=<Did
  Name> RuntimeSettings!DidModule=<Did Module Name>
  ```

Request runtime settings per level

If you request a list of the configuration runtime settings for a particular level, you will see the displayed information on the runtime settings types that deviate from the default for the given level. If no level parameters like `RuntimeSettings!Environment`, `RuntimeSettings!Did`, or `RuntimeSettings!DidModule` are used, the settings that deviate from the default are listed for a global level.

Parameter	Description
`List!Configuration=RuntimeSettings`	This parameter gives an overview of all settings on the global level that deviate from the default.
`List!Configuration=RuntimeSettings` `RuntimeSettings!Environment=Published`	This combination of parameters results in an overview of all settings on the `Environment=Published` level that deviate from the default.

`List!Configuration=RuntimeSettings`
`RuntimeSettings!Environment=Published`
`RuntimeSettings!Did=LCLDID.ENG`
	This combination of parameters results in an overview of all settings on the `Environement=Published` and `Did=LDLDID.ENG` levels that deviate from the default.

`List!Configuration=RuntimeSettings`
`RuntimeSettings!Environment=Published`
`RuntimeSettings!Did=LCLDID.ENG`
`RuntimeSettings!DidModule=LCLSRC.ENG`
	This combination of parameters results in an overview of all settings on the `Environment=Published`, `Did=LDLDID.ENG`, and `DidMudle=LCLRSC.ENG` levels that deviate from the default.

Request runtime settings per setting type

If you enter a specific setting type, you get an overview of all settings for the given setting type, including the default values and the level where these settings are set.

Parameter	Description
`List!Configuration=RuntimeSettings` `RuntimeSetting!settingType=General`	This combination of parameters results in an overview of values for all general settings, including the default values on the global level.

`List!Configuration=RuntimeSettings`
`RuntimeSetting!settingType=XMLFile`
`RuntimeSettings!Environment=Current`
	This combination of parameters results in an overview of values for all XMLFile settings, including the default values on the current environment level.
	For each setting, you can define if it has the default value, the value set on the global level, or the value set for the environment level.

Set values for configuration settings

To change the value of the configuration settings type, use the following command line:

```
ConfigureInstanceSettings <Configuration Type Name>!<Setting Name>=<Value>
```

You can set more than one setting. When changing settings, you cannot request help or view a list of the configuration settings.

Note The runtime settings and the configuration settings cannot be set at the same time.

Set values for runtime settings

To change the value of the runtime settings type, use the following command line:

```
ConfigureInstanceSettings <Setting Type Name>!<Setting Name>=<Value>
```

You can set more than one setting. When changing settings, you cannot request help or a view list of the runtime settings.

Note The runtime settings and the configuration settings cannot be set at the same time.

For the runtime settings, you can also specify a level where the setting will be set. If no specification is provided, the setting will be set at the global level.

- Use the `RuntimeSettings!Environment` parameter to indicate the environment to set the settings for.
- Use the `RuntimeSettings!Did` parameter to indicate the DID to set the settings for.
- Use the `RuntimeSettings!DidModule` parameter to indicate the DID module to set the settings for.

For more information, see Runtime settings levels.

Remove runtime settings

As the runtime settings can be stored at various levels, you can remove any of these settings. As a result, the value for this setting is used at another level according to the runtime settings levels logic, or the default value is used. For more information, see Runtime settings levels.

You can remove a setting using the following command line:

```
ConfigureInstanceSettings RuntimeSettings!Remove=<Setting Type>!<Setting>
```

Note The notations `<Setting Type>` and `<Setting>` are also accepted.

Only one setting can be removed at a time. You cannot remove a setting and also request help or a list of the runtime settings at the same time. For the runtime settings, you also can specify a level with the setting to remove. If no specification is given, the setting is removed at the global level.

The runtime settings parameters and their usage:

- Use the `RuntimeSettings!Environment` parameter to indicate the environment to remove the setting for.
- Use the `RuntimeSettings!Did` parameter to indicate the DID to remove the setting for.
- Use the `RuntimeSettings!DidModule` parameter to indicate the DID module to remove the setting for.

Use the ManageInstanceEnvironment tool

In KCM Core, the runtime settings are organized in a hierarchy, where the following levels exist:

- Global level.
- Environment level.
- DID level. The database interface definitions levels located within an environment.
- DID module level. These levels are located within a DID. Each DID has at least one default DID module. DID modules represent connection types that are used in templates.

Note For more information about DIDs and DID modules, see the *Kofax Communications Manager DID Developer's Guide*.

The runtime settings can be defined at each level. A setting that is assigned a value at one level can be overridden by a value at another level according to the following hierarchy:

- If no value is found at the DID module level, the value at the DID level is used.
- If no value is found at the DID level, the value at the environment level is used.
- if no value is found at the environment level, the value at the global level is used.
- If no value is found at the global level, the default value is used.

Use the ManageInstanceEnvironment tool to manage the environments, DIDs, and DID modules for KCM Core. With this tool, you can create your own environments, DIDs, and DID modules. Once these elements are created, you can use the ConfigureInstanceSettings tool to manage the runtime settings for them.

Environments

For environments, the following functionality is available.

Parameter	Description
`Environment!List=true`	Use this parameter to see an overview of the environments and to define which environment in the list is the default one.
`Environment!Add=<environment>`	Use this parameter to add a new environment.
`Environment!Remove=<environment>`	Use this parameter to remove an environment.
`Environment!Default=<environment>`	Use this parameter to set a default environment.

DIDs

A DID is available for all environments. The below table lists the parameters and their functionality for DIDs.

Parameter	Description
`Did!List=True`	Use this parameter to see an overview of DIDs. The displayed list includes DIDs added as the DID files in the KCM Core DID folder.

`Did!Add=<Did Name>`
`Did!DidModule=<Did Module Name>`
`Did!ConnectionType=<Connection Type>`
	To add a new DID, enter the DID name using the `Did!Add` parameter and specify its connection type with the `Did!ConnectionType` parameter.
	The `Did!DidModule` parameter is optional. This parameter specifies the name of the default DID module of this DID. If you do not enter this parameter, the default module will be created with the same name as this DID itself.

`Did!Remove=<Did Name>`	Use this parameter to remove a DID. **Note** DIDs added as the DID files in the KCM Core DID folder cannot be removed using this parameter. To do so, remove the DID file from the KCM Core DID folder manually. The KCM Core DID folder is a subfolder of the KCM Core Workfolder.

DID modules

The DID modules represent connection types and are available per DID. Each DID contains at least one DID module.

Parameter	Description
`DidModule!List=True`	Use this parameter to receive a list of all DID modules per DID.
`DidModule!Did=<Did Name>`	Use this parameter to indicate the DID for which DID modules are managed: listed, added, or removed.

`DidModule!Add=<Did Name>`
`DidModule!Did=<Did Name>`
`DidModule!ConnectionType=<Connection Type>`
	To add a new DID module to a DID, enter the DID module name using the `DidModule!Add` parameter.
	To indicate a DID to which the DID module is entered, use the `DidModule!Did` parameter.
	Also, provide the `DidModule!ConnectionType` parameter for the DID module.

`DidModule!Remove=<Did Module Name>` `DidModule!Did=<Did Name>`	To remove a DID module from a DID, enter the DID module name using the `DidModule!Remove=<Did Module Name>` parameter and the DID with the DID module to remove using the `DidModule!Did=<Did Name>` parameter.

Use the RestartServices tool

To apply the changes made in KCM settings on Docker, use the RestartServices tool. This tool restarts the KCM Core document processors and can be used both for the Contract Manager and KCM instance containers.

You can use optional parameters to restrict the number of affected services or even limit the action to stop particular services only. If you execute the RestartServices tool without any arguments, all KCM services of the KCM installation are stopped. Then, all KCM services, regardless of whether they were stopped before the command or not, are started again.

The following table lists the RestartServices tool parameters to perform the restart.

Parameter	Description
`Restart!ContractManager=<True or False>`	Optional. Default value is `True`. Indicates whether the KCM Contract Manager services are affected. This parameter has no impact on the KCM instance setup since no KCM Contract Manager services are present. On Docker, this parameter must be omitted. It is provided for future compatibility with non-Docker installations.
`Restart!Instances=<True or False>`	Optional. Default value is `True`. Indicates whether KCM instance services are affected. This parameter has no impact on the KCM Contract Manager setup since no KCM instance services are present. These services comprise the KCM Repository Server, the Content Management API services, and the KCM Core services. On Docker, this parameter must be omitted. It is provided for future compatibility with non-Docker installations.
`Restart!DpsOnly=<True or False>`	Optional. Default is `False`. Indicates that only the KCM Core document processor services are affected. Sets up both the `Restart!ContractManager` and `Restart!Instance` parameters to `False` to ensure no other services are affected. This parameter has no impact on the KCM Contract Manager setup since no KCM Core document processor services are present.
`Instance!Number`	Optional. Default value is `All` if no `Instance!Number` parameter is given. Indicates the instance to restart services for. `Instance!Number=All` parameter indicates all instances available in the setup. This parameter has no impact on the KCM Contract Manager setup since no KCM instance services are present. On Docker, this parameter must be omitted. It is provided for future compatibility with non-Docker installations.
`Restart!StopOnly=<True or False>`	Optional. Default is `False`. Indicates services to be stopped and no services to be started.

Manage language packs on Docker

In the Docker environment, managing language packs is similar to a regular installation.

To get a language pack for a particular language, follow these steps.

1. To get a language pack for a particular language, on the Contract Manager container start the Windows PowerShell command prompt or use the direct "docker exec" command.
2. Execute the following script script.
   `Add/Remove/Get/List-LanguagePack(s).ps1`

   Note Ensure that the script is executed with the management folder as a working directory.

Examples

Below is an example of the `ListLanguagePacks.ps1` script running on the Docker container, using the Windows PowerShell interactive session on the host.

```
docker exec cm_container powershell -Command 'CD "C:\KCM\Programs
\5.4\Management"; .\ListLanguagePacks.ps1'
```

On the instance containers, the scripts are replaced with the executables that perform the same purpose, except the extensions. They are renamed in the same way and accept the same parameters.

Below is an example of the `ListLanguagePacks.psl` script running on the instance container.

```
docker exec instance_container ListLanguagePacks.exe
```

Other management tools on Docker

Other management tools documented for a regular installation are either replaced with an executable variants with the same name, behavior and parameters, or are not relevant on Docker.

Note In the Docker environment, the management tools generally do not require the `Instance! Number` parameter.

See the list of the replaced tools with the identical executable variants on the instance containers:

- ListPublications
- PurgePublications
- RollbackPublication
- CheckOutputManagementMetadata
- ClearOutputManagementMetadata
- SaveOutputManagementMetadata
- LoadOutputManagementMetadata

The `SetCMAuthentication.ps1` tool functions as a regular on Docker. Ensure that this tool is executed with the management directory as a working directory.

The following management tools are not relevant on Docker:

- Any tools for Output Management (services, content, and other tools)
- Uninstall, Upgrade, Activate, ListCCMPackages, Add/Remove-Instance