Kofax Communications Manager

ComposerUI for ASP.NET Developer's Guide

Version: 5.3.0

Date: 2019-05-28

KOFAX

Table of Contents

Preface

This guide describes the structure and configuration of KCM ComposerUI for ASP.NET along with an overview of related components.

Related documentation

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

https://docshield.kofax.com/Portal/Products/CCM/530-1h4cs6680a/CCM.htm

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

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

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 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 offline access, see the "Product documentation" section in the *Installation Guide*.

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

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

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.

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

KCM ComposerUI Server is a server-based solution to produce interactive documents. It consists of three main components: KCM Core, KCM Repository Server, and KCM ComposerUI. Based on KCM Core, KCM ComposerUI is a scalable, high-performance solution for advanced browser-based correspondence and business document systems. It supports end users in creation of complex correspondence and production of highly personalized documents. KCM ComposerUI has a browser-based user interface that can be integrated with any business application and data source.

Browser-based solution

KCM ComposerUI supports the creation of interactive documents. Users can generate documents from a list of available document templates. This is a highly interactive process, when users answer the additional questions in the forms opening in their browsers.

These forms:

- Capture data or text that is not available from existing applications
- Enable users to make specific choices from predefined options
- Can dynamically adapt themselves to previous answers or application data

KCM ComposerUI automatically creates all these forms based on instructions in the Master Template. No knowledge of HTML is required. The forms support the use of data entry fields, check boxes and drop-down menus. Help texts assist the end user in the process. All dialog formatting is done using CSS and XSL style sheets, which enable full customization of the web form layout.

Result document

After completing the interview process, the Master Template creates the required document. Depending on the KCM ComposerUI setup, the user can edit the document in a word processor. KCM ComposerUI can also convert the document to PDF format, and then email or publish it.

Chapter 2

Overview

Interactive documents produced by KCM ComposerUI require additional user input for document production. During the server-based production of the document, the user may be prompted for additional information.

KCM Core is the foundation of KCM Document Production. KCM Core can access a wide range of data sources and allows a seamless integration of the document production process in the business processes of an organization. KCM Core runs as a number of Windows services that satisfy all kinds of standard or custom requests. KCM Core is designed to be highly scalable, and it is accessible through a number of APIs. See *KCM Core Developer's Guide* for more information on these APIs.

KCM document production is template-based. In KCM terminology a template is called a "Master template." KCM Repository offers an environment in which these models can be developed in a controlled fashion. It provides role-based authorization, revision management, reporting facilities, and dependency information. Also, dynamic objects such as Text Blocks and dynamic Forms, can be managed in KCM Repository. When producing a document that requires a Text Block or a dynamic Form, KCM Core retrieves these at runtime from the KCM Repository Server. See KCM Designer online help for more information on Text Blocks and Forms.

KCM ComposerUI is a web-based component that serves as an intermediary between KCM Core and the user. Whereas KCM Core communicates in terms of XForms XML messages, HTML Forms have to be presented to the user. Likewise, whereas the browser will post data in HTTP messages, KCM Core requires answers in XML format. KCM ComposerUI takes care of the translation between these two components. It is highly configurable and may be customized by adapting all kinds of resources, such as cascading style sheets, xsl transformations or resource files.

The chapter "Sample workflow" demonstrates that KCM ComposerUI can be used as a stand-alone application, which presents a list of Master Templates and supports document production given a selected Master Template. In many cases document production is offered to the user as part of another application. Therefore, KCM ComposerUI has been designed to integrate into another application easily.

This process involves three parties:

1. Integrating application, which can be anything from a simple client(-server) application to a complete portal.
2. End user
3. KCM ComposerUI

KCM ComposerUI offers different interfaces:

1. A server-to-server interface to allow the integrating application to prepare a Master Template run in which a single interactive document is produced. Preparation of a Master Template run involves a set of parameters and optionally one or more data files. A Master Template run is uniquely identified by a session identifier.

2. An HTTP interface to allow the end user to produce a document given such a session identifier. The user is guided through a wizard-like sequence of HTML forms.
3. A server-to-server interface to allow the integrating application to access the results of a Master Template run. For example, the produced document and some metadata about the Master Template run.

If the definition of the Master Template run is simple, it can also be passed as part of the HTTP interface. In most relevant cases, though, server-to-server preparation of Master Template runs is preferred.

See Integration for more details on integration of KCM ComposerUI in another application.

KCM ComposerUI consists of two components: An ASP.NET implementation that runs on Microsoft Internet Information Server, and a J2EE implementation that runs on J2EE-based web servers.

Installation

KCM ComposerUI requires installation of KCM Core in combination with either the ASP.NET or the J2EE implementation of KCM ComposerUI that depends on your web server. To take advantage of dynamic objects, such as Text Blocks or dynamic forms, an installation of the KCM Repository Server is required as well.

See *KCM Installation Guide* for details on products installation.

Each implementation of KCM ComposerUI comes with two examples of how to configure and customize this application. A more detailed examples of applications to explore the product will be given In the next chapter.

Summary

- KCM Core offers production of interactive documents through the KCM ComposerUI API, with all the advantages of a server-based solution.
- During this process the KCM Repository Server may serve dynamic objects such as Text Blocks and dynamical forms.
- KCM ComposerUI is a web-based component that operates as an intermediary between KCM Core and user.
- KCM ComposerUI offers interfaces to allow for seamless integration in another application.

Chapter 3

Sample workflow

This chapter includes a sample workflow that demonstrates the generic functionality of KCM ComposerUI and shows how it can be configured and customized. Some information in this chapter may vary, depending on your implementation of KCM ComposerUI. In these cases, information for ASP.NET is provided with its equivalent for J2EE in brackets. Most information applies to both the ASP.NET and the J2EE implementations.

Before working with KCM ComposerUI, you should configure Letterbooks and Document Templates using KCM Designer. In the following example, assume that a Letterbook *Examples* has been created.

After installation and configuration of a Letterbook in the KCM Designer, you can configure a default Letterbook entry point in the configuration page of your application using: `http://[ machine ]:[ port ]/itp/app/[ application ]/configure.aspx`. For more information, see Application configuration.

You can also pass the Letterbook entry point as a parameter like this:

http://[machine]:[port]/itp/app/sample/modelselect.aspx?letterbook=[letterbook]

where:

- [machine] is the name of the web server
- [port] is the port through which it is accessible
- [Letterbook] is the letterbook entry point you want to use

When you have entered a default Letterbook on the application configuration page, it is not necessary to pass the Letterbook parameter on the page. If you do pass a Letterbook parameter, it will override the default setting on the configuration page. Name your Letterbook *Examples*. The following page appears:

This page shows one of the two main functionalities of KCM ComposerUI, namely the possibility to retrieve and present a structured list of Document Templates that has been defined as a Letterbook in KCM Designer. The list shows all the models with the name *Examples*, which are available in the Letterbook for the user.

Depending on the [Letterbook] the Administrators can manage the content of the Letterbook as follows.

The value of the Letterbook parameter can be provided as a rep:/ URI. It looks similar to this example:

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

host: TCP/IP Hostname of the system hosting the KCM Content Publication Database / KCM Repository server.

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

type: Type object to retrieve. Supported objects:

- `Letterbook`

project: Project. Use asterisk symbol to refer to the default project.

path: (Optional) Folders, separated by slash.

object: The object to be retrieved.

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

- `user`=Repository User
- `status`=[published|accepted|current|development]

Example:

`rep://localhost/letterbook/DemoProject/Letters?status=accepted`

refers to the Accepted version of the Letterbook `Letters` in the project `DemoProject`. The Letterbook is retrieved from the KCM Repository Server installed on the local host. See RetrieveRepositoryObject for more information on REP:/ URIs.

Note Letterbook is not aware of environments. The settings for ContentPublicationName and RepositoryObjectStatus as configured in the Environments are not applied.

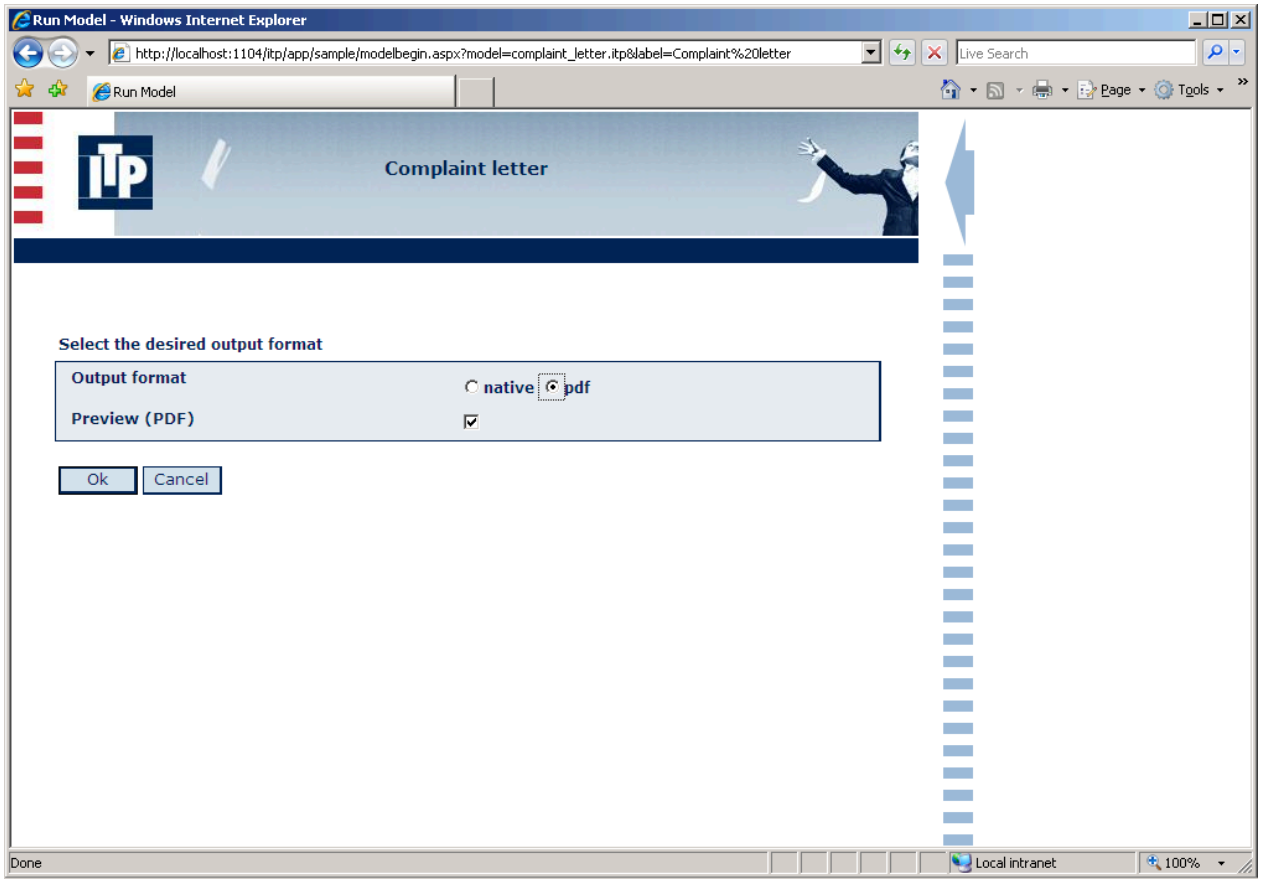
When no rep:/ URI is provided in the [letterbook], the templates (named Logical Models in the KCM Base Administration GUI) added to the Letterbook are managed using the KCM Base Letterbook GUI.

The page consists of four frames. The two outer frames serve as margins. The left inner frame shows a (possibly nested) list of folders. From this list, a page with the Master Templates for the currently selected folder is loaded in the right frame.

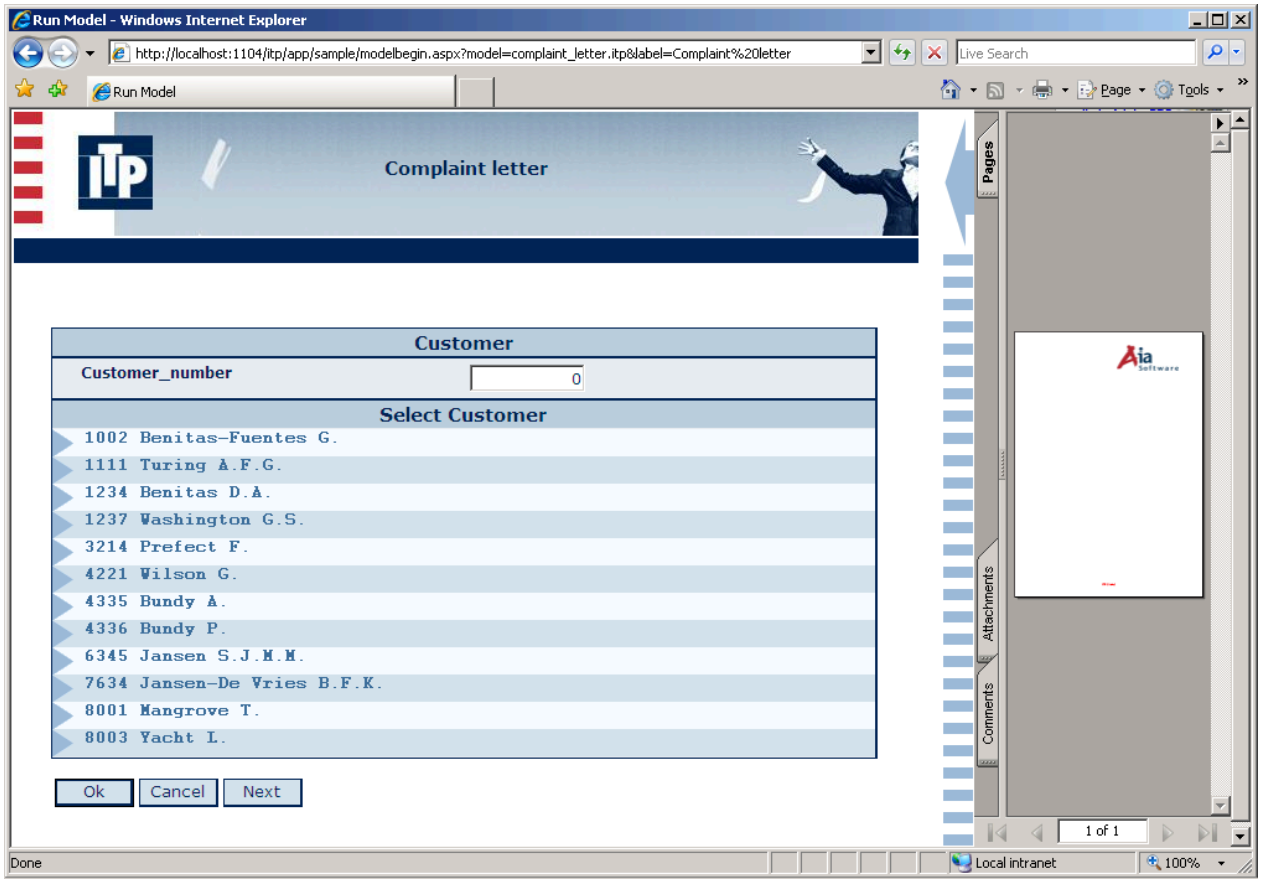
In the right frame, Master Templates are presented as links. Click the link "Complaint letter" to open the following URL in a new browser window:

http://[machine]:[port]/itp/sample/modelbegin.aspx

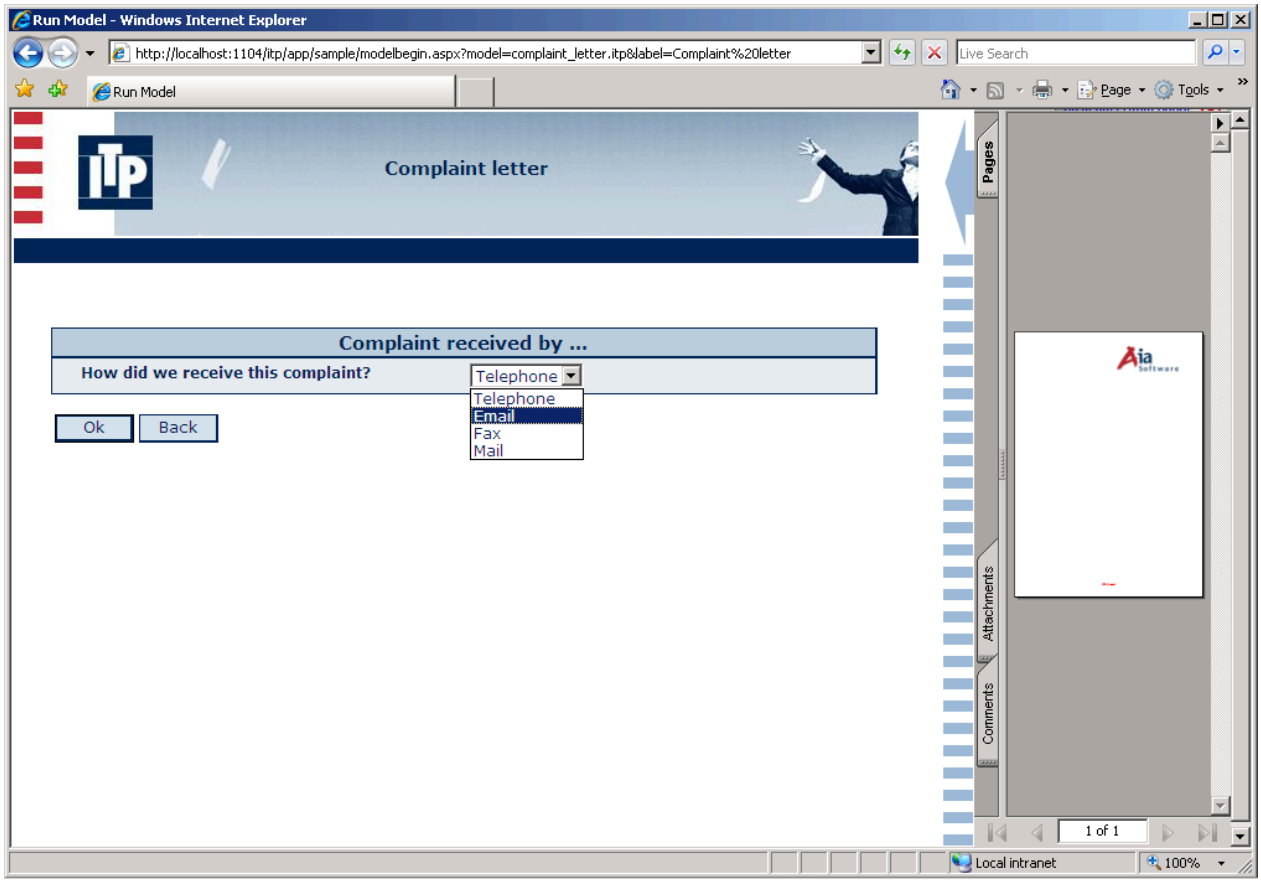
This URL is extended with a query string, which contains information about the model to be run.



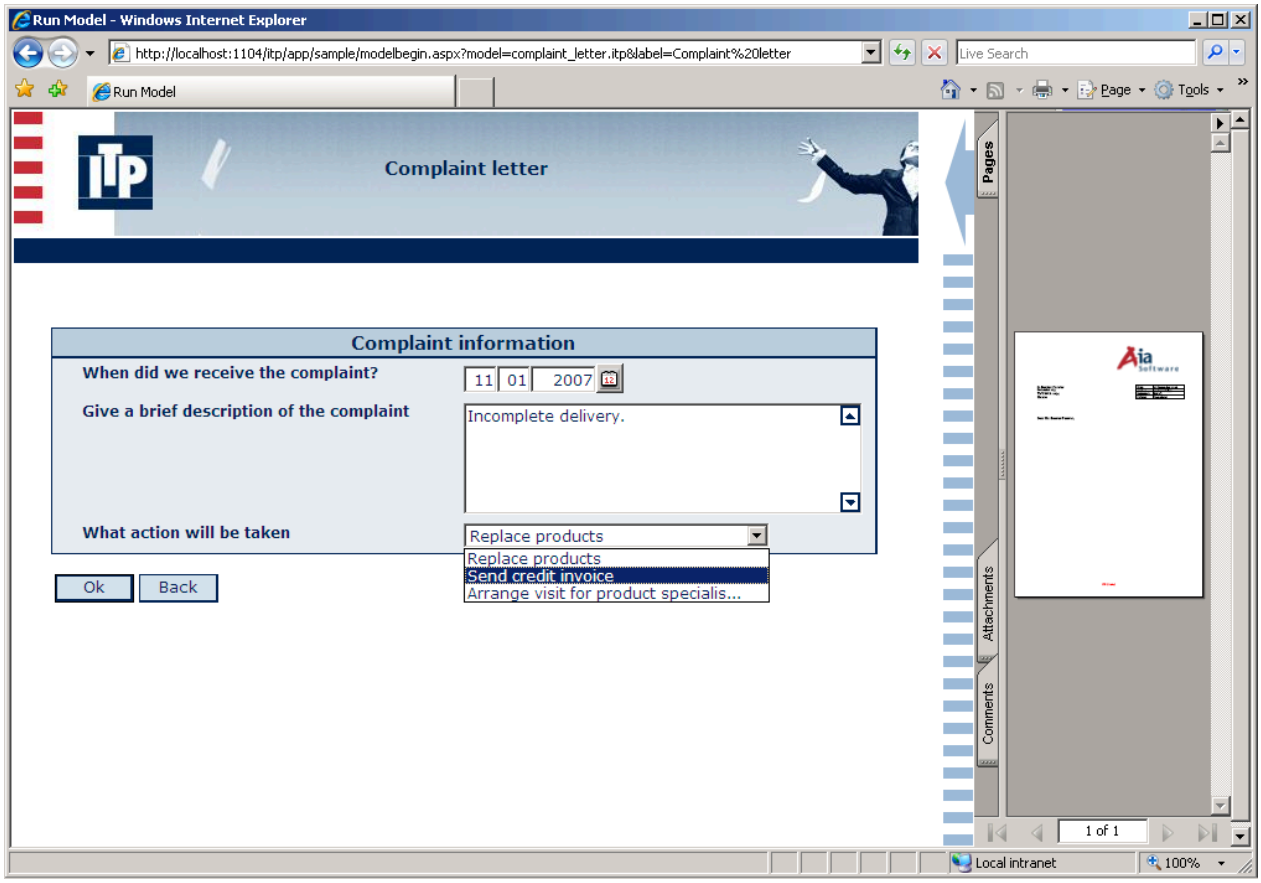
When the page appears, the user can select an output format and Preview preference. Whenever KCM sends out additional questions, it can include them with the document in its current stage of production. The preview document may be presented to the user. Leave the default settings as they are. Click OK to start the Master Template run on the server.



The next page shows the first form of the Master Template run. The Complaint letter Master Template requires selection of the customer to whom the letter should be sent. KCM Core will send out an XForms XML message translated by KCM ComposerUI to an HTML page. In the right frame, a preview of the document is shown, which contains only a logo. Click "1002 Benitas-Fuentes G." to proceed.



The second form contains a drop-down list with the medium options. A user should select the option through which the complaint was received. No additional content is produced and the preview remains the same. Select Email and click OK.



In the last form the user is prompted to provide a date, a brief description of the complaint, and an action to take a response to the complaint. Now the document has a header containing some of the information provided earlier: the name of the customer and the fact that the complaint was received by email. The information provided in the previous form is reflected in the preview. Provide answers to the questions and click OK.

The Master Template run is now completed. The resulting PDF document appears in the right frame. Click the blue arrow to resize the frame. Now browse to

http://[machine]:[port]/itp/app/sample2/modelselect.aspx?letterbook=[Letterbook]

where:

- [machine] is the name of the web server
- [port] the port through which it is accessible
- [Letterbook] indicated the Letterbook entry point you want to use

This leads to similar, but slightly different features:

- The appearance of the pages is different.
- The page asking for the document format and preview activation is not available.
- No previews are shown and the result document is presented in Microsoft Word format.

Analysis

Both runs were started by opening a URL in a following format:

```
http://[machine]:[port]/itp/app/[application]/modelselect.aspx or
http://[machine]:[port]/itp/app/[application]/modelselect.aspx?
letterbook=[letterbook] or (http://[machine]:[port]/itp/app/[application]/
modelselect.jsp or http://[machine]:[port]/itp/app/[application]/
modelselect.jsp?letterbook=[letterbook]),
```

where:

- [machine]:[port] refers to the web site with installed KCM ComposerUI
- [application] identifies the usage of "Application"
- [letterbook] indicates the usage of the Letterbook entry point

An application identifies a set of resources, that are relevant to the configuration and customization of OnLine, such as configuration settings, web pages and cascading stylesheets. Browsing to different applications results in the use of different resources. Therefore, in different appearance and behavior.

Browsing to this URL resulted in a list of Master Templates. When the user clicked any Master Template another URL of the form was opened:

```
http://[machine]:[port]/itp/app/[application]/modelbegin.aspx?[querystring]

(http://[machine]:[port]/itp/app/[application]/modelbegin.jsp?[querystring]),
```

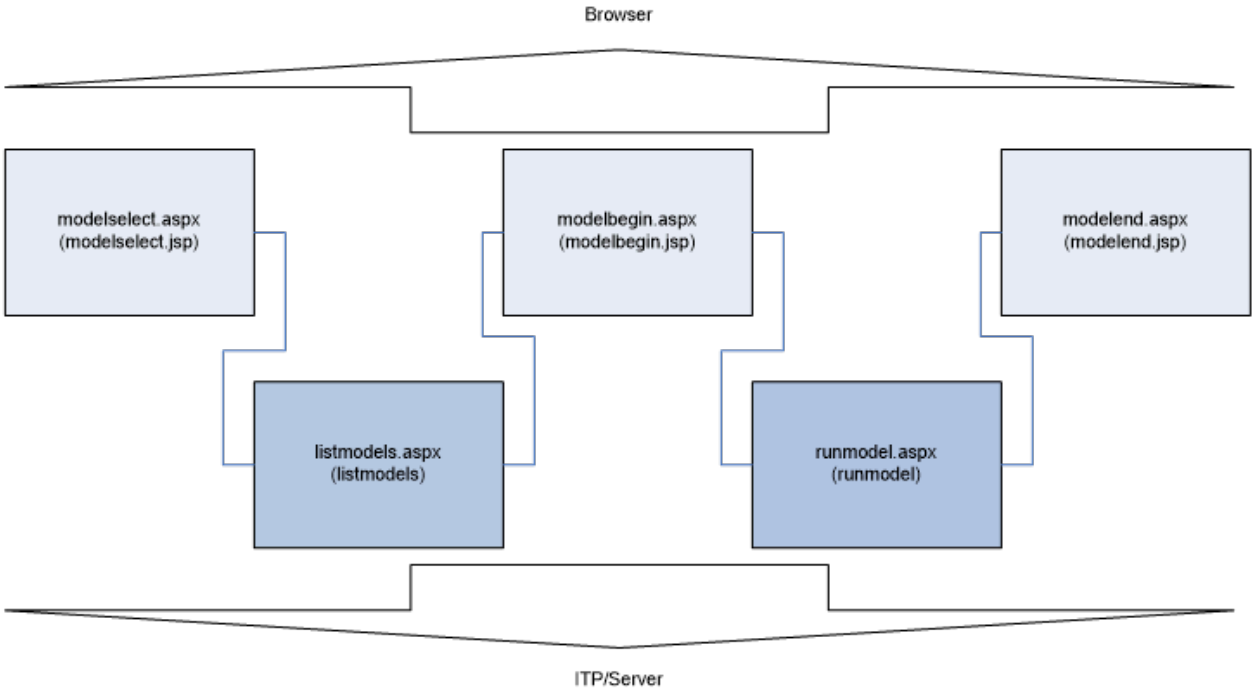
where [querystring] contained some information about the Master Template to be run. This initiated the actual Master Template run, possibly after some additional pages. During the Master Template run a number of forms were presented to the user, after which a URL was opened

```
http://[machine]:[port]/itp/app/[application]/modelend.aspx?[querystring]

(http://[machine]:[port]/itp/app/[application]/modelend.jsp?[querystring])
```

The [querystring] contained some additional information about the document that was produced, allowing the modelend page to open the result document.

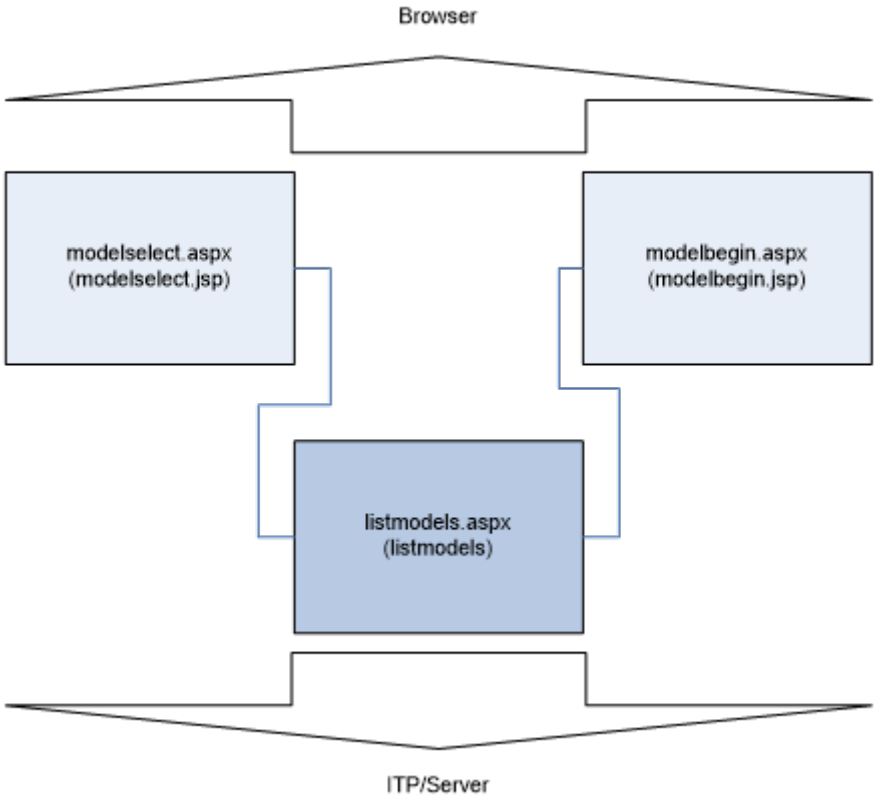
The process flow described in the previous paragraphs is described here:



The modelselect, modelbegin and modelend pages are the specific component between the generic KCM ComposerUI functionality offered by listmodels and runmodel.

Modelselect, modelbegin and modelend are part of the application resources. Typically, they build up framesets, show additional pages (like the page in sample, asking for document format and preview), open result documents (modelend) and support application integration.

Both listmodels and runmodel are considered as separate generic "calls" embedded in a specific preceding and succeeding page. In many cases the runmodel functionality will be used by directly opening the modelbegin page with a number of query string parameters.



Summary

There are two main functionalities offered by KCM ComposerUI:

- **listmodels** available to the user
- **runmodel**, which interactively produce a document by running a model

These functionalities are always called in the context of an Application. An application identifies a set of resources that determine the appearance and behavior of KCM ComposerUI. Listmodels and runmodel are always embedded in preceding and succeeding pages, which are part of an application.

Chapter 4

Configuration

Use the Main and Application configuration pages to:

- Set up KCM ComposerUI
- Configure the number of general settings
- Maintain (create, delete) applications
- Get access to a specific configuration page for each application

Main configuration

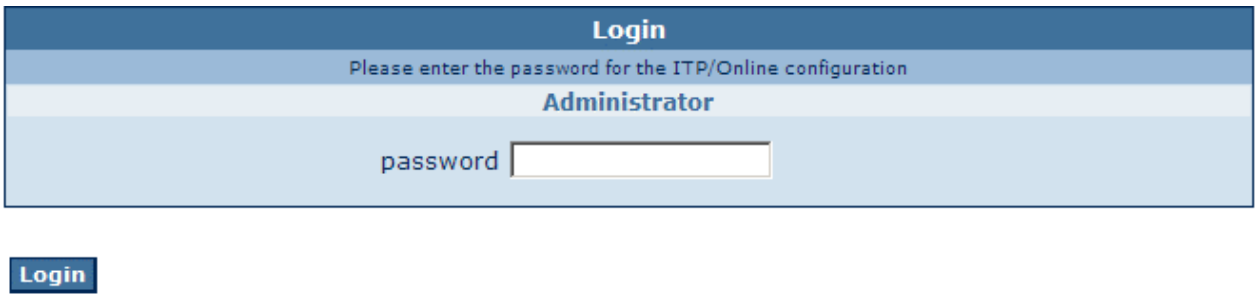
Browse to `http://[machine]:[port]/itp/configure.aspx (http://[machine]:[port]/itp/configure)`.

The Main configuration page consists of three sections:

- Administrator section
- KCM Core section
- Applications section

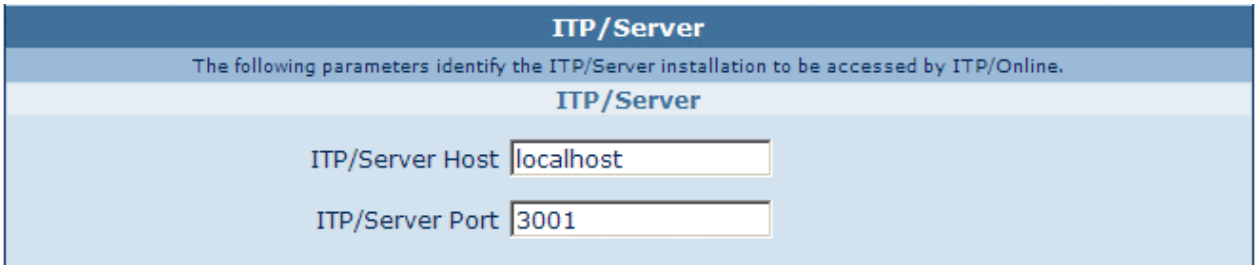
Administrator section

Use the Administrator section to set the user password that gives access required to access configuration pages. When the user enters the password, the login page will appear. The configuration page appears after the user logs in.



CM Core section

Use the KCM Core section to configure connection parameters



- **KCM Core Host**

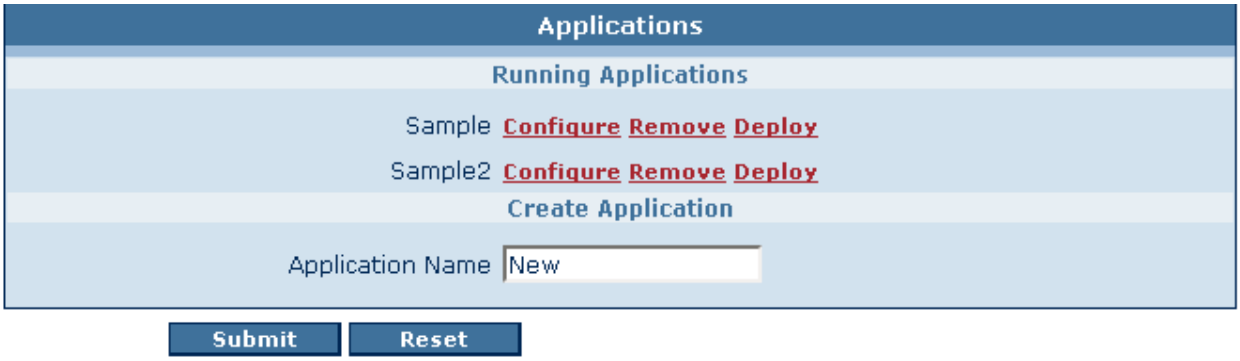
  Name of the host running KCM Core. This name is used in the communication between KCM ComposerUI and KCM Core.
- **KCM Core Port**

  Port number of the KCM Core host. Used in the communication between KCM ComposerUI and KCM Core. You can find the port number of your KCM Core installation on the DP Manager tab in the KCM Core Administrator.

Application section

The application section includes all applications available to the KCM ComposerUI installation. Here you can create applications, remove applications and access the specific configuration page of each application.

Two implementations of KCM ComposerUI, J2EE and ASP.NET, differ from each other according to the changes in their Application directory. See Applications. In the J2EE implementation changes take effect immediately, whereas in the ASP.NET implementation it is required to take an explicit deployment step. For that click "Deploy" link in the Applications section of the configuration page. This link is only available in the configuration page of the ASP.NET implementation of KCM ComposerUI.



In this example, the Sample and Sample2 applications are already running. To configure these applications, click Configure, or to remove them, click Remove. Removal of the application does not delete the directory and configuration files that belong to the application, but removes only the application from the administration. To restore the application, add an application with exactly the same name as the removed one.

To add a new application, fill out the Application Name input box. In the example above, the application called "New" is created by clicking the button Submit. After its creation, the new application still needs to be configured and customized. For more information, see Application configuration and Customization.

Note When using Microsoft IIS 7, the deploy process may take a very long time (up to an hour), because of the slow access to the IIS 6 compatibility metabase. If a timeout error occurs during deployment, you can continue the process by starting the deploy process again from the main configuration page. The KCM ComposerUI log file will contain an informative message once the deployment has finished successfully.

When using Microsoft IIS 7, the deploy process may take a very long time (up to an hour), because of the slow access to the IIS 6 compatibility metabase. If a timeout error occurs during deployment, you can continue the process by starting the deploy process again from the main configuration page. The KCM ComposerUI log file will contain an informative message once the deployment has finished successfully.

Such long loading times are expected in the following cases:

- Creation of a new KCM ComposerUI application.
- Deploy one of the KCM ComposerUI sample applications (Sample, Sample2, and SecureSample).
- KCM ComposerUI application change from non-secure mode to secure mode, or from secure mode to non-secure mode.

Application configuration

Click Configure to set up the application configuration page. The application configuration page consists of the following sections:

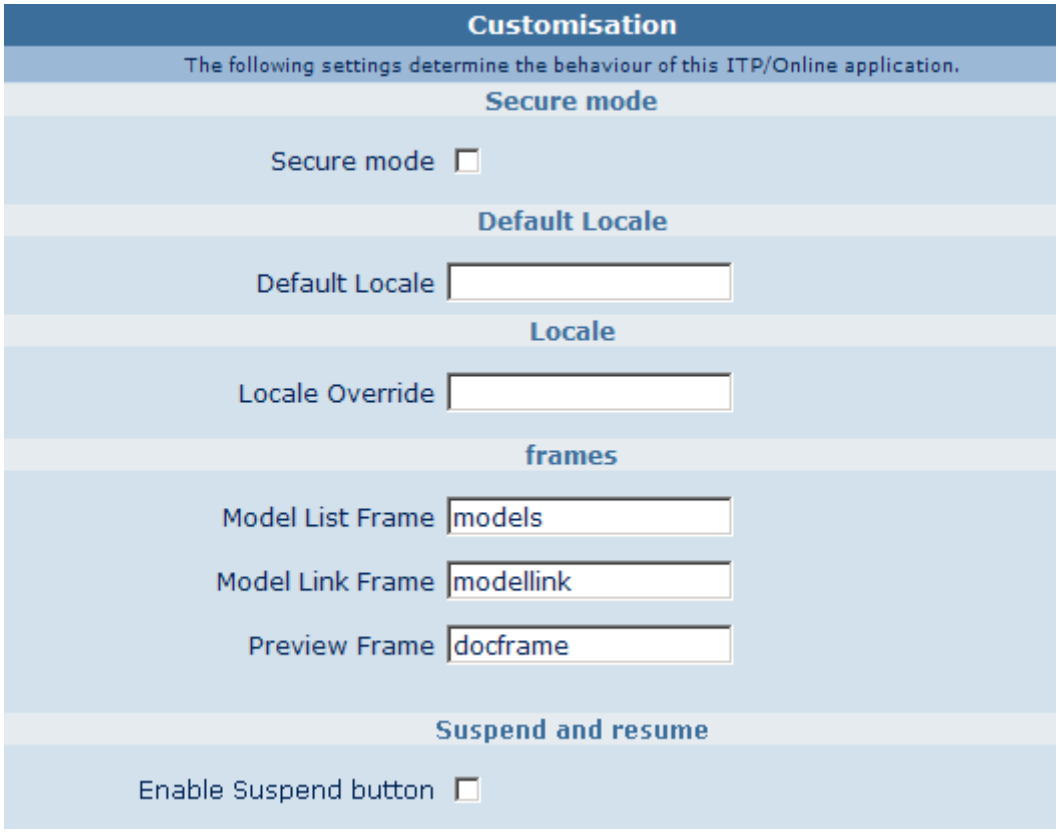
- Main Configuration section
- Customization section
- Properties section

Main configuration section

To open the main configuration page click "Configure" link in the main configuration section.

Customization section

The Customization section presents settings to determine the KCM Designer interaction with the KCM ComposerUI application.



Secure mode

Select the check box "Secure mode" to make Secure mode available in the KCM ComposerUI application. See Securing CM ComposerUI for more information on how to use this functionality.

Default Locale

To determine the language of the user interface KCM ComposerUI uses the Language Preference of the browser. If the configured language is not supported by your application installation, KCM ComposerUI uses the language assigned to the Default Locale.

By default, only two languages are supported in KCM ComposerUI: "en" (English) and "nl" (Dutch). To support other languages, add your own language files. See Customization for more information.

Locale Override

If a language is configured for the Locale Override, KCM ComposerUI ignores the language preference of the browser and uses the locale override instead.

Master Template List Frame

Master Templates in the Master Templates list frame are presented as links. To open the page Modelbegin in another frame referenced by a name, and click the link. This name is configurable as the "Master

Template Link Frame". If a frame with this name does not exist, or if it equals the value _blank, the page Modelbegin is opened in a new browser window.

Preview Frame

The requested previews are shown in a frame, referenced by a name. This name is configurable as the "Preview Frame". If a frame with this name does not exist, or if it equals the value _blank, the preview will be opened in a new browser window.

Enable Suspend button

Select the check box "Enable Suspend" in the KCM ComposerUI configuration screen to make Suspend and resume functionality available. See Suspend and resume for more information on how to use the Suspend and resume functionality.

Properties section

To open the pages in the chapter Sample workflow, a number of parameters are passed. Default settings of some of these parameters, so called properties, can be configured. If such a parameter is not passed to a page, its configured value is used. On the application configuration page, each of these properties is described with its corresponding call.

Summary

- KCM ComposerUI can be configured from the browser
- Access to the configuration pages may be protected by a password
- There is one main configuration page and one configuration page per application
- On the application-specific configuration pages, defaults may be configured for some parameters of Listmodels and Runmodel.

Configuration for CM ComposerUI ASP.NET

KCM ComposerUI ASP.NET edition exposes some configuration settings that are not available through the generic configuration pages. Instead, these settings are found in the file web.config, which is located in the IIS virtual directory of KCM ComposerUI ASP.NET. The folder location of the virtual directory is configured during the installation. In a default installation it is C:\Inetpub\wwwroot\itp. The file web.config is an XML document, and the KCM ComposerUI ASP.NET configuration settings are stored in the XML element `<appSettings>`. The default configuration is shown here:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <appSettings>
    <add key="itp_log_dir" value="C:\ITPOnLineApps\itplog" />
    <add key="itp_temp_dir" value="C:\ITPOnLineApps\sessiondata" />
    <add key="itp_applications_dir" value="C:\ITPOnLineApps" />
    <add key="itp_log_count" value="10" />
    <add key="itp_max_log_size_kb" value="1024" />
  </appSettings>
<!-- ... -->
</configuration>
```

Each line of the form `<add key="configuration setting" value="value">` defines the value of a single configuration setting. The following table describes the settings:

Settings	Description
itp_log_dir:	• The folder where KCM ComposerUI ASP.NET stores its log files. In a default installation, this is the subfolder itplog of the KCM ComposerUI application folder. • Note that if this setting is changed to a different folder, then the permissions described in the *KCM Installation Guide* must be applied to that folder too. It is also recommended to remove the permissions from the old folder when they are no longer needed.
itp_temp_dir:	• The folder where KCM ComposerUI ASP.NET stores its temporary files and its session information. In a default installation, this is the subfolder *sessiondata* of the KCM ComposerUI application folder. • Note that if this setting is changed to a different folder, then the permissions described in the *KCM Installation Guide* must be applied to that folder too. We also recommend to remove the permissions from the old folder when they are no longer needed.
itp_applications_dir:	• The KCM ComposerUI applications folder. Subfolders of this folder contain the sources of the custom applications installed in KCM ComposerUI ASP.NET. • Note that if this setting is changed to a different folder, then the permissions described in the *KCM Installation Guide* must be applied to that folder too. It is also recommended to remove the permissions from the old folder when they are no longer needed.
itp_log_count:	The maximum number of old log files that KCM ComposerUI ASP.NET retains.
itp_max_log_size_kb:	The size that an KCM ComposerUI ASP.NET log file may reach before it is archived as an old log file.

Job scheduling in CM Core

In KCM Core, jobs are distributed over the available KCM Document Processors in the order they arrive. If a single KCM Core installation is used to serve both KCM ComposerUI Server and batch jobs, long-running batch jobs may cause a delay in the processing of the interactive KCM ComposerUI Server requests. To prevent this, KCM Core offers two options that cause KCM ComposerUI jobs to be run before other jobs. Add one or both of these to the dp.ini file in the Config folder of the ITPWork folder, and restart KCM Core.

```
[Configuration]
PrioritizeOnLine=Y       ; Y or N, default value N
DedicatedOnLine=<number> ; Number of CCM Document Processors to
                         ; reserve, default value 0
```

When set to Y, PrioritizeOnLine causes KCM Core to schedule KCM ComposerUI jobs ahead of other jobs, if taking jobs from the queue. Note that this only has an effect if both KCM ComposerUI jobs and other jobs are queued. If all KCM Document Processors are processing jobs and a KCM ComposerUI job is submitted, it has to wait until a KCM Document Processor has finished processing its job.

With DedicatedOnLine you can reserve one or more KCM Document Processors for KCM ComposerUI jobs. These KCM Document Processors are not used for other, possibly long-running, jobs. This makes it much more likely that KCM Document Processor will be available quickly for processing of KCM ComposerUI Server job, if one is submitted.

Chapter 5

Calls

The Listmodels and Runmodel functionalities are considered as "Calls", because they both have a number of parameters.

The behavior of each call in KCM ComposerUI depends on a large set of parameters. Overall, there are three types of parameters:

1. `Call` parameters are call-specific parameters that vary at run time. They can only be passed as a parameter of the Call.
2. `Configuration` parameters are installation-specific parameters that are not vary at run time. They can only be configured through the configuration pages.
3. `Properties` are parameters that may or may not vary at run time. For properties, defaults may be configured on the application specific configuration page. If a property is not passed as a parameter on the Call, the configured default is used.

The following parameter types are supported:

- **String**
- **Number** (string that may be interpreted as a number)
- **Boolean** (string that may be interpreted as a boolean value, both 'Y'/'N' and 'true/false' are accepted, either in upper or in lower case)

All parameter names are case insensitive.

Call parameters

Parameters can be passed in a call in two ways:

1. `Get` method, as part of the query string
2. `Post` method, as form parameters

Both methods may be mixed in a single call. The call functionality does not depend on the method used to pass the parameter. If a parameter is passed both as part of the query string and as a form parameter, the latter will prevail.

Note When Secure Mode is enabled for an application, the form parameters are ignored. The parameter `sessionid` of the Runmodel call can only be passed through the query string.

Configuration parameters

Configuration parameters can only be configured through the main configuration page or the application-specific configuration pages.

Main configuration page:

```
http://[ machine ]:[ port ]/itp/configure.aspx
```

Application-specific configuration page:

```
http://[ machine ]:[ port ]/itp/app/[ application ]/configure.aspx
```

Properties

To limit the amount of parameters that need to be passed in a call to KCM ComposerUI, the concept of a "Property" is introduced. A Property is a Call parameter with the possibility of default configuration.

Listmodels

To see a nested list of models go to:

```
http://[ machine ]:[ port ]/itp/app/[ application ]/listmodels.aspx
```

```
(http://[ machine ]:[ port ]/itp/app/[ application ]/listmodels)
```

A call to listmodels will result in two HTML pages, one automatically loading the other. The first page shows a nested list of folders. The latter is loaded in the Model List Frame and shows a list of models for a selected folder in the list of folders.

We recommend that you provide a frameset, which contains a frame with the name configured as the Model List Frame. This can be done in the modelselect page:

```
http://[ machine ]:[ port ]/itp/app/[ application ]/modelselect.aspx
```

```
(http://[ machine ]:[ port ]/itp/app/[ application ]/modelselect.jsp)
```

See CM Core: OnLine exit points for more information on how the list of models is retrieved from KCM Core.

Parameters (prepared model list)

If a model list has already been prepared by the integrating application, a single `sessionid` parameter is sufficient to start the model list. For more information, see Integration. Other parameters will be ignored.

Call parameters

Parameter	Type		Description
sessionid	string	required	The session identifier corresponding to the prepared model list. This session identifier can only be passed through the query string, not as form data.

Parameters

If the sessionid parameter is omitted, the following property is used to identify the Letterbook.

Properties

Parameter	Type	Description
Letterbook	string	• The entry point of a Letterbook. This selects the folders and Templates that are available for the user. This entry point is interpreted as the name of an interactive Letterbook, optionally followed by subfolders separated by slashes (/). • Starting with KCM Core and KCM Repository version 4.2.3y when a rep:/ uri is provided for the Letterbook, the Letterbook will be retrieved from the KCM Repository. See also RetrieveRepositoryObject.

Note

- The root and pattern properties have been deprecated. Use the Letterbook property instead.
- To allow root parameters to function, the ListModels.exe executable that is used to retrieve the Letterbook has to be configured to produce legacy references. See the chapter "Post-installation steps" of *KCM Installation Guide*.

Runmodel

A call to this page initiates a model run based on a number of passed parameters. The model that is run may contain FORM statements, possibly referring to dynamic Forms. If it does, this will result in a sequence of HTML pages containing forms being presented to the user.

Parameters (prepared model run)

If a model run is already prepared by the integrating application a single sessionid parameter is sufficient to start the model run. Other parameters ignored. See Integration chapter.

Call parameters

Parameter	Type	Status	Description
sessionid	string	required	• The session identifier corresponding to the prepared model run. • This session identifier can only be passed through the query string, not as form data.

Parameters

If the sessionid parameter is omitted, the following parameters are used:

Call parameters

Parameter	Type	Status	Description
model	string	required	Identification of the model to be run given as a rep:/ URI or Letterbook URI. See also Rep:/URIs.
label	string	optional	User readable model identification.
keys	string	optional	Semicolon-separated list of keys to be used in the model run.

Parameter	Type	Status	Description
extras	string	optional	Semicolon-separated list of extras to be used in the model run.
res_uri	string	optional	• The location of the result document as a local path relative to either the web server (res_srv=N) or the KCM server (res_srv=Y). See the Properties table below for a description of the res_srv property. • The default is an empty string in which case KCM ComposerUI stores the documents on the Web Server and makes it available for display in the browser. In that case the document is deleted when the session expires.
pvw_proc_params	string	optional	• A string that passes to the ProcessPreview exit point. This can be used freely to pass information to the exit point. For example, to specify which action it should take. • For more information, see CM Core: Online exit points.
res_proc_params	string	optional	• A string that passes to the ProcessResult exit point. This can be used freely to pass information to the exit point. For example, to specify which action it should take. • For more information, see CM Core: OnLine exit points.

Properties

Parameter	Type	Description
dat_srv	Boolean	The XML data (if applicable) can be accessed from the server on which KCM Core is running. If this is not the case, the file identified by dat_uri (if any) is uploaded from the web server to the KCM server. In this case the value "*DataURI" should be passed for the keys parameter, thus indicating the file that was uploaded to the KCM server.
dat_uri	String	Location of the XML data to be used (if any), as a local path relative to either the web server (dat_srv=N) or the KCM server (dat_srv=Y).
db_pwd	String	Password to be used by the DataManager on KCM Core.
db_uid	String	User Id to be used by the DataManager on KCM Core.
env	String	Identification of the Environment under which the model is run on KCM Core. This corresponds to the connection configuration on the server.
history	Boolean	An outline of the provided answers to all interacts should be returned by KCM Core at the end of the model run.
ofcmd (AS/400 only)	String	OnFailure command, executed when the model fails.
oscmd (AS/400 only)	String	OnSuccess command, executed when the model succeeds.
postcmd (AS/400 only)	String	Post command, executed at the end of the run (after ofcmd or oscmd)
precmd (AS/400 only)	String	Pre command, executed at the start of the run.
pvw	Boolean	KCM ComposerUI HTML Forms should be accompanied with a preview of the document as it is at that moment.
pvw_fmt	String	Required format of the preview document (if any - either native or PDF).
res_fmt	String	Required format of the result document (either native or PDF).

Parameter	Type	Description
res_owt	Boolean	The file identified by res_uri may be overwritten.
res_srv	Boolean	KCM Core should write the document to the location identified by res_uri. If a res_uri is provided, res_srv=Y makes sense in most cases. If a res_uri is not provided, res_srv is ignored, the document is stored in local storage on the web server and be made available through a URL.

Chapter 6

Applications

The Application concept was initially introduced In the chapter Sample Workflow. Application defines a set of resources that determine the appearance and behavior of KCM ComposerUI.

Application folder

An application is stored on the file system of the web server as subdirectory of the Applications folder. The sample application that is shipped with KCM ComposerUI is implemented as a *newsample* folder.

Each application folder may contain a number of files and subdirectories that define the appearance and the behavior of the application. For most of these files, KCM ComposerUI defines defaults. Whenever a file is available in the Application folder, it replaces the default.

At the root level of the application folder the configuration settings are stored in a file called config.xml. The contents of this file can be manipulated through the configuration page of the application. For more information, see Configuration. The root level may also contain web pages (aspx-pages for ASP.NET or jsp pages for J2EE). Each of these pages will be accessible through the following url:

```
http://[machine]:[port]/itp/app/[application]/[page]
```

where:

- [machine] is the name of the web server
- [port] the port through which it is accessible
- [application] is the name of the application
- [page] is the name of the page

The pages produced by KCM ComposerUI refer to a cascading stylesheet in:

```
http://[machine]:[port]/itp/app/[application]/css/styles.css
```

This means that a styles.css file may be stored in the css subdirectory to determine the appearance of the pages.

The resources subdirectory may contain files related to localization of texts.

The xslt subfolder may contain overrides of the xslt files that determine the structure of the html produced by KCM ComposerUI.

Defaults

The default value is used If a resource is not available in the application folder. These defaults are installed as part of KCM ComposerUI. When overriding a default resource, the default resource itself can be a useful basis to start from.

The location of default resources differs between the ASP.NET and the JBoss implementation of KCM ComposerUI:

- **ASP.NET**
  The default resources can be found in the physical path of the virtual directory of KCM ComposerUI (referred to by the installer as "virtual directory physical path"). The default is C:\inetpub\wwwroot\itp.
- **J2EE**
  The default resources can be found in the itp.war file that is contained in the itp.ear file of KCM ComposerUI.

Customization

Applications can be customized on the following levels:

1. Customizing fonts and colors. See Styles for more information.
2. Customizing the behavior of the application. See Behavior for more information.
3. Customizing text of the user interface. See Text for more information.
4. Defining structure. See Customizing XSLT for more information.

The easiest way to create a customized application is to:

1. Create the application and its folder in the general configuration page.
2. Copy the files of one of the sample applications that are shipped with the KCM ComposerUI installation to the folder of the newly created application. Refer to KCM Installation Guide for more information.
3. Adapt the files of the newly created application as described in the following paragraphs.

In the J2EE implementation of KCM ComposerUI, customization in the application folder is applied immediately (or automatically). In the KCM ComposerUIASP.NET implementation, it is necessary to deploy the custom application. Application deployment is done through the main configuration page of KCM ComposerUI ASP.NET.

Styles

The technique of application styles customization is based on creating and adapting stylesheets. In order to customize application styles, you should create a CSS directory in the application configuration directory and put the following files there:

- styles.css stylesheet contains the definitions for the KCM interact pages.
- textblock.css stylesheet contains definitions for the Text Block preview window. Only needed if you actually use Text Blocks.

The Sample application that is shipped with KCM ComposerUI provides CSS files example. Images used by these stylesheets should be placed in an "images" subdirectory inside the CSS directory. No defaults are available for these resources.

Behavior

In addition to changing styles, you can also change the behavior of an application. This customization is based on defining .aspx (.jsp) pages that precede or succeed the basic listmodels and runmodel functionality. These pages could also define frames in which particular information is shown.

As it was mentioned in the Sample workflow chapter, there are three "exit point" pages. If you place the following files in the root of the application directory, they will override standard variants of KCM ComposerUI.

- **modelselect.aspx (modelselect.jsp).**

  This page acts as the starting point of the application. You can place any file at the root of the application config and invoke that via the /app/ path. The default modelselect.aspx (modelselect.jsp) just calls listmodels.
- **modelbegin.aspx (modelbegin.jsp).**

  This page is called after selecting a model, before it starts to run. The default page just calls runmodel.
- **modelend.aspx (modelend.jsp).**

  This page is called after running a model. The default one opens the produced document in the frame "docframe" or in a new window if you have not defined such a frame.

The default exit point implementations use a number of predefined pages that you can call in your own exit points as well. You can also override these pages so that the default exit points use them instead. The predefined pages are:

- empty.aspx (empty.jsp) shows an empty page, using the right style
- opendocument.asxp (opendocument.jsp) opens a document asynchronously, that is shows a "loading..." text while the document loads
- openfolders.aspx (openfolders.jsp) calls listmodels asynchronously, and shows a "loading..." text while retrieving the list

The defaults for these pages can be found in the jsp subdirectory of the KCM ComposerUI installation. When overriding them, they should be placed at the top level of the application directory.

In the implemented pages aspx or jsp, it is possible to use various bits of functionality offered by KCM ComposerUI. The APIs that are available to the customized pages are described in The CM ComposerUI Server customization APIs.

Behavior examples

- **Framesets**
  You can use the exit points not only to define behavior, but also to define framesets. In the application configuration page you can specify the frame, where certain interactions should occur. For example, to have the folder and models opened in two frames in the same window, you could have the following modelselect.aspx (modelselect.jsp):

```
<HTML>
 <HEAD>
  <TITLE>ITP/OnLine Server Sample Application</TITLE>
 </HEAD>
 <FRAMESET cols="250,500" framespacing="0" scrolling="no">
  <frame src="openfolders.jsp" name="folders" />
  <frame src="empty.jsp" name="models"/>
 </FRAMESET>
</HTML>
```

  If the value "models" is configured as Model List Frame on the configuration page of the application, the list of models is now shown in the right frame.
- **Open Microsoft Word non-maximized**
  The modelend.aspx (modelend.jsp) exit point for the provided Sample2 and newSample applications use ActiveX to open the result document in Microsoft Word. By default, the window in which Microsoft Word is opened is maximized. For ITPOLSActiveX3 this behavior can be changed by locating the doInitWordControl() function, and adding wordcontrol.MaximizeUponPopup = 0 to it. This leads to the following block of code:

```
function doInitWordControl()
{
   wordcontrol.OLSUploadURL = "";
   wordcontrol.OLSDownloadURL = "";
   wordcontrol.FileAccessStyle = 1;
   wordcontrol.Language = "en";
   wordcontrol.MaximizeUponPopup = 0;
}
```

Text and JavaScript behavior

Part of the text that you see on screen, such as interact questions, originates from the KCM Master Template itself. Some of the text is defined in KCM ComposerUI. Localized versions of these text parts are stored in the subfolder "resources." You can override the standard text for a particular language by creating a "resources" folder in your application containing the following files:

- **<lang>_custom.msg**, this file contains the text like header and footer text, titles and "supporting" interact text, such as "loading document..."
- **<lang>_errors.msg**, this file contains the text of the error messages
- **<lang>_gui.msg**, this file contains the text that is shown in the configuration pages
- **<lang>.js**, this file contains the text that are used by JavaScript code that is executed at the client, such as the calendar control

For the new div-oriented KCM ComposerUI output, additional <lang>.js resource files have been placed in the subfolder "res". This language file may also contain code that sets the texts of the jQueryUI DatePicker, which is used in the new output. For more information, see res\nl.js and http://docs.jquery.com/UI/Datepicker/Localization.

Some text, like "Ok" and "Back", originates from KCM Core. When the TranslateOnLineResources setting on KCM Core is set to "N", KCM Core will output message codes instead of literal texts. For the div-oriented output, these messages are translated using the resource files in the subfolder "res". See res\nl.js for more information.

The TranslateOnLineResources on KCM Core can be found on the General tab of the Environments.

The behavior of the pages is determined by two javascript variables:

- **numEditFormat**

  This variable determines whether numerical fields are shown in one or in two input boxes. In <lang>.js, assign 1 to this variable to show only one input box for a numerical field. The default value is 2, in which case two input boxes will be shown: one box on each side of the decimal point.
- **fileEditActiveX**

  It is a boolean variable that determines whether or not an ActiveX-control is used for file upload questions. By default this is the case (fileEditActiveX = true). If the variable is set to false, a native HTML file upload control is used. In this case, the default value for the file upload question, the default provided in the model, is ignored. For the div-oriented output this variable is deprecated. Use the variables in res\settings.js instead.

The Language Preference settings of your browser and the Default Locale and Locale Override settings of your application determine which files KCM ComposerUI will use. The defaults for these resources can be found in the folder "resources" of the KCM ComposerUI installation. When overriding them, they should be placed in a subfolder "resources" inside the application folder.

Customizing XSLT

With XSL stylesheet overrides, the user can change the structure of the different pages. To do so, we recommend to create a folder XSLT, that contains the following XSLT files:

- html.xsl, describes the general structure of the HTML pages.
- modellist.xsl, is applied to the XML, where the modellist structure of the Models directory is shown. This stylesheet produces the list of folders as well as the list of models that belong to a certain folder
- error.xsl. Applied to the XML that contains the error returned from the API. It produces an error page.
- interact.xsl, produces the HTML FORMS in which the user can fill in the questions as they have been defined in the KCM Model. This XSL is applied to the xForms XML that is returned by the Start and Continue method.

The defaults for these XSL stylesheets can be found in the "xslt" subdirectory of the KCM ComposerUI installation. When overriding them, they should be placed in an "xslt" subdirectory inside the application directory.

Note XSL stylesheets can only be adapted by specialists experienced in writing XSL stylesheets. Furthermore, the KCM ComposerUI stylesheets will be updated in future updates of KCM ComposerUI. As a result, you may need to update your own stylesheet, or create a new custom stylesheet.

Chapter 7

Suspend and Resume

KCM ComposerUI offers Suspend and Resume functionality. Users can suspend a Master Template run without losing answers that are already completed, and then resume it a later point in time. See Customization section.

With Suspend and Resume functionality each KCM ComposerUI form, which runs within the configured application, contains a button Suspend at the bottom.

Note KCM Core
KCM Core 3.2.21 or higher. It is only available when using prepared Master Templates runs, as described in chapter Integration.

Note If your KCM Core version supports the DisableValidation environment setting, do not use it. Disabling validation will prevent the correct functioning of Suspend and Resume.

Default Suspend implementation

When the user clicks the Suspend button, KCM ComposerUI will submit all answers currently filled in and forward the user to the page modelsuspend.aspx (modelsuspend.jsp). The modelsuspend page calls the KCM Core script, which stores the Master Template run session. Then sends it back to the modelsuspend page, which offers the user a downloadable file, containing this suspended session.

Both the page modelsuspend.aspx (modelsuspend.jsp) and the script KCM Core SuspendSession are marked as exit points. For example, their implementation may be altered to store the suspended session on a server. Or, instead of offering it as a download to the user, pass the suspended session back to the calling application. See CM Core: OnLine exit points.

Default Resume implementation

KCM ComposerUI comes with a page called modelresume.aspx (modelresume.jsp) to resume a stored session. In the default implementation, this page asks the user for a file containing a stored session, which

is uploaded to KCM Core. KCM Core uses the file to restore it in a new KCM session, which is used to resume the stored session. Next, the user is forwarded to KCM ComposerUI form, which was originally shown when the Suspend was performed.

Both the modelresume.aspx (modelresume.jsp) page and the ResumeSession script are marked as exit points, so they can be modified. For example, instead of asking the user to upload a file, these exit points could let KCM Core retrieve the stored session file.

KCM Core also contains the exit point script SessionResumed, which is called after a suspended session is restored. See CM Core: OnLine exit points.

Changing Forms during suspension

KCM ComposerUI Server supports changes to Forms while the model runs using the suspended Forms. The same applies to KCM Master Templates. Answers to the Questions of Forms are validated by KCM Core when a Master Template run is resumed. This validation checks all previous answers against the current definition of each Form that was already filled in before suspending the Master Template run. If the validation check fails for one or more Forms, the Master Template run will start at the first Form in the Master Template that fails the validation.

The validation consists of checks if the Form ID is changed, if any Questions are changed, and if all answers comply with the restrictions defined in their corresponding Question. When the validation fails, the Form is presented again with appropriate error messages about the Questions that no longer accept the previous answers. Most of the Questions retain their previous input, or in case of a new Question, their default answers. After changing the KCM Master Template its developer should consider forcing some of the Forms to be presented again. We recommend that you change the Form IDs or Question IDs when resuming a model run.

Forms are only considered unchanged when their IDs are specified and unchanged. Otherwise, the Form fails the validation and is presented again. The order of the Forms in the Master Template is not important, they are matched to their ID and the number of times that it was presented during the Master Template run.

For more information on Form IDs, refer to the *KCM Core Developer's Guide*, chapter Form and question IDs.

Questions are considered unchanged, when:

- Their IDs are identical
- The type of the Questions is not changed
- The answer is still valid for that Question.

The order of Questions is not important because they are matched to their ID. New Questions with a new ID added to the Form are considered as changed. Removed questions are considered as unchanged. Changing a Question from a single-select to a multi-select or vice versa also constitutes a change of the Question type. For more information on Forms and Question definitions, see *Template scripting language Help keyword FORM.*

Answer is valid if it respects the maximum length and specified format in the definition of the Question. If the Question specifies a list of possible answers, the answer must occur in that list. A Text Block selection

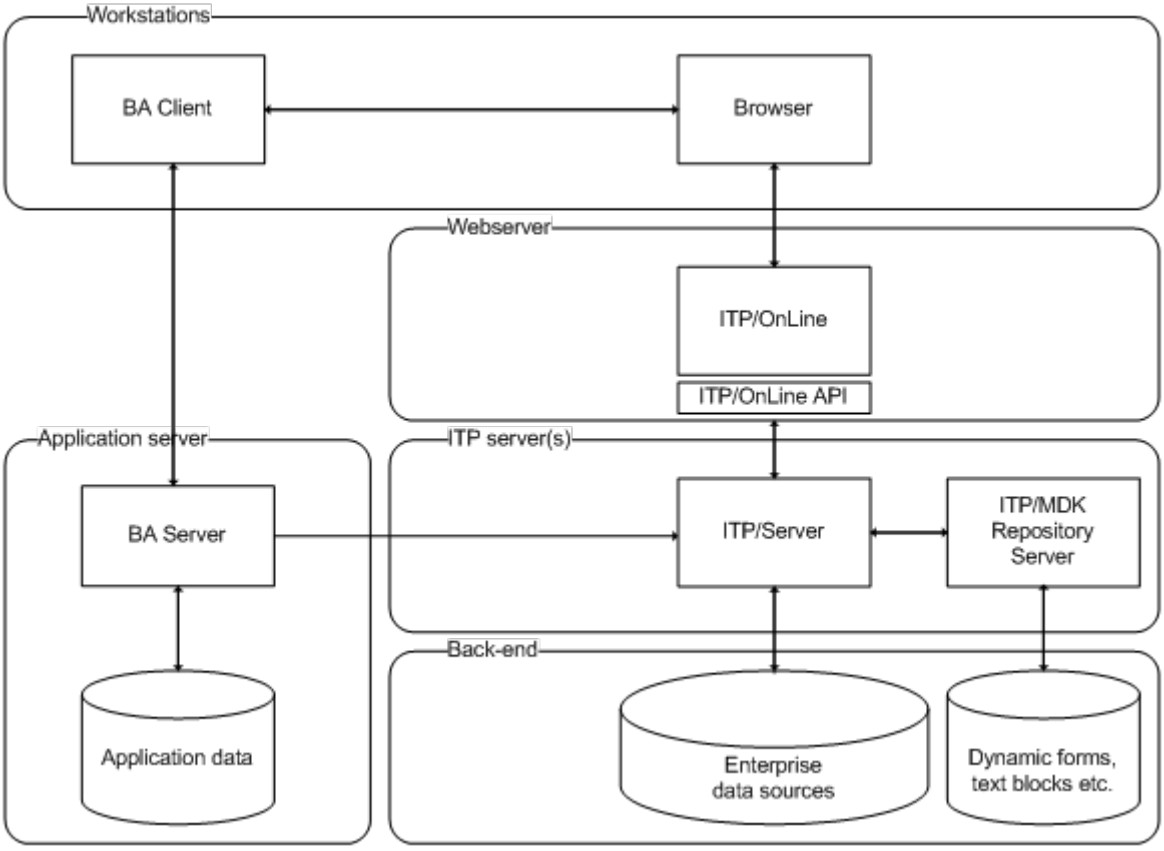
must also be a member of the specified View, in order to be a valid answer. For more information on Question definitions, see *Template scripting language Help keyword FORM.*

Chapter 8

Integration

In most of the cases KCM ComposerUI Server is used within the context of integrating application. Business applications often can not produce documents in a proper fashion due to lack of functionality. KCM ComposerUI fills this gap.

The following diagram shows a situation when a business application requires a document to be produced. This business application consists of a client and a server component, named "BA Client" and "BA Server," respectively. Note that this is a logical architecture that may be mapped on a physical architecture in many different ways.



KCM ComposerUI and business application have two interfaces between them:

1. An http interface between KCM ComposerUI on the web server and the BA Client on the workstation, supporting client-side integration of Master Template runs.

2. A server-to-server interface between KCM Core and the BA Server, supporting server-side integration of Master Template runs.

Integration of an interactive Master Template run involves a number of steps:

- Preparation of the Master Template run. This involves a number of calls from the BA Server to KCM Core. See Server side integration.
- The Master Template run itself, consisting of a sequence of HTML Forms, offered by the http interface between KCM ComposerUI and the BA Client. See Client side integration.
- Processing of the Master Template run. This can be achieved by specific implementations of some exit points on KCM Core. See Server side integration.

The fact that a single Master Template run consists of multiple steps introduces the notion of a session, which is shared by these steps.

Note Business application integration using prepared model is introduced in KCM ComposerUI 3.2.20. Although the current version of KCM ComposerUI can be used in combination with earlier versions of KCM Core, the integration method described in this chapter is not available in such configurations. Instead, one should use the Runmodel call for non-prepared model runs, which is described in chapter Calls. This chapter also describes model list integration using prepared model lists. This functionality is in KCM ComposerUI 3.2.25. When an KCM Core version 3.2.24 or earlier is used, one should use the call Listmodels for non-prepared model lists, which is also described in chapter Calls.

Sessions

In KCM ComposerUI, the sessions that hold the information about a single Master Template run are open. They can be accessed from any KCM Core script that runs within the context of a session:

- Key/value pairs can be stored in a session and retrieved at a later time; any script can do this, as long as it runs in the context of the specific session.
- The parameters that define KCM ComposerUI Master Template run are stored as special key/value pairs in the session. See *KCM CoreScripting Language Guide* for more information on sessions, which use a KCM Core concept and are not specific to KCM ComposerUI.

Client-side integration

Client-side integration focuses on the embedding the document production process in the flow of the BA Client. The exact way in which it can be achieved highly depends on the nature of the client application. In general, the following occurs:

- The BA Client requests the BA Server to prepare a Master Template run. This results in the creation of a session defining the Master Template run. The BA Server will responds with a URL, which contains the session identifier as a query string parameter.
- The BA Client opens this URL in a separate browser, in an iframe on a web page or in a browser component. This triggers an interactive Master Template run.

- As soon as the end user completes the Master Template run, the modelend page of the KCM ComposerUI application is opened. The particular implementation of this page determines what happens next. For example:
  - Opening the result document, so the end user can edit it.
  - Calling a service on KCM Core for further processing of the document. This can be done in combination with any of the other options.
  - Redirection to a page in the business application, putting the business application back in control, if the business application is web-based.
  - Redirection to a page with a specific name. The BA Client can detect that the Master Template run is finished, if the business application is a Windows application and the URL is loaded inside a browser control.
  - Simply closing the browser, if the URL is loaded inside a separate browser window.

Server-side integration

Server-side integration focuses on the embedding the document production process in the business logic of the BA Server. It consists of two parts:

1. Preparation of the Master Template run. A number of KCM Core services are available for this purpose.
2. Notifications, that can be implemented using exit point scripts. See CM Core: OnLine exit points.

Note The integration methods described here are not compatible with all versions of KCM Core. See introduction of the Integration chapter for more information on the supported KCM Core version.

Prepare Master Template list

In KCM ComposerUI, Master Templates list functionality can also be prepared through the server-to-server interface.

Preparation of a Master Template list consists of the following steps:

1. Create a new session for the Master Template list.
2. Set the parameters that define the KCM ComposerUI Master Template list.
3. Optionally perform preparation for a subsequent Master Template run. For example, this could uploading data files or setting Master Template parameters.

KCM Core offers a service for each of these steps and services that combine these steps. See Preparation services for details on the services for preparing Master Template lists and Master Template runs.

It is also possible to extend the preparation of the Master Template list so that custom information can be stored in the session. This can be done by creating a custom service, which stores additional key/value-pairs in the session, and calling this service as an additional step. This information becomes available from all scripts that run in the context of the session. For example, the information is available during the exit point scripts or subsequent Master Template run.

Preparation of a Master Template list results in a session identifier. This session identifier can be used to invoke the actual Master Template list by opening the URL from the BA Client:

```
http://[ host ]:[ port ]/itp/app/[ application ]/modelselect.aspx?
sessionid=[ sessionid ]
```

See section Parameters (prepared Master Template list) for more information.

Prepare Master Template run

Preparation of a Master Template run consists of the following steps:

1. Create a new session for the Master Template run.
2. Optionally, store files in the session. This could for example be an XML file holding the data based on which the Master Template should be run.
3. Set the parameters that define the KCM ComposerUI Master Template run.

KCM Core offers a service for each of these steps. Additionally, it offers a service that combines these steps to a single call. Finally, it also offers a service to retrieve a file from a session.

It is also possible to extend the preparation of the Master Template run so that custom information can be stored in the session. This can be done by creating a custom service, which stores additional key/value pairs in the session, and calling this service as an additional step. This information becomes available from all scripts that run in the context of the session, such as the exit point scripts.

Preparation of a Master Template run results in a session identifier. This session identifier can be used to invoke the actual Master Template run by opening the URL from the BA Client:

```
http://[ host ]:[ port ]/itp/app/[ application ]/modelbegin.aspx?
sessionid=[ sessionid ]
```

See section Parameters (prepared Master Template run) for more information.

Preparation services

The KCM Core services for the preparation of Master Template runs or Master Template lists are described below. Apart from ITPOLSSessionStart, these services should be run in the context of an existing session. This can be done by passing the identifier of this session on the call to the service. Details on how to do this depend on the specific API used to call the service. See the documentation on the APIs in the chapter "Integrating" of the *KCM Core Developer's Guide*. For more information on the role of sessions in KCM Core scripts, see the chapter "KCM Core Sessions" in the *KCM Core Scripting Language Developer's Guide*.

ITPOLSSessionStart

Parameters

```
(no parameters).
```

ITPOLSSessionStart creates a new session. After a call to the service, the session identifier is returned.

ITPOLSSessionUploadFile

Parameters

```
Parameter Text FileName
```

Through ITPOLSSessionUploadFile, a file may be uploaded to an existing session. On the parameter FileName, the name of the file is passed. This name may not contain backslashes. The service calls ReceiveFile with this name as the Src parameter and stores the received file in the session. If a file with this name already exists in the session, it is overwritten.

ITPOLSSessionDownloadFile

Parameters

```
Parameter Text FileName
```

Through ITPOLSSessionDownloadFile, a file may be downloaded from an existing session. On the parameter FileName, the name of the file is passed. This name may not contain backslashes. If no file with this name exists in the session, an error appears.

ITPOLSSessionSetModelParams

```
Parameter Text User
Parameter Text Model
Parameter Text Label = ""
Parameter Text Keys = ""
Parameter Text Extras = ""
Parameter Text ResultFileName = "Result"
Parameter Text ProcessPreviewParams = ""
Parameter Text ProcessResultParams = ""
Parameter Text ProcessConfirmParams = ""
Parameter Text Environment = ""
Parameter Text ResultFileFormat = ""
Parameter Text Preview = ""
Parameter Text PreviewFileFormat = ""
Parameter Text Confirm = ""
Parameter Text ConfirmFileFormat = ""
Parameter Text DatabaseUID = ""
Parameter Text DatabasePWD = ""
Parameter Text OnSuccessCommand = ""
Parameter Text OnFailureCommand = ""
Parameter Text PreCommand = ""
Parameter Text PostCommand = ""
Parameter Text DBB_XMLInput = ""
Parameter Text DBB_XMLOutput = ""
Parameter Text RedirectUrl = ""
```

The parameters defining a Master Template run may be set through ITPOLSSessionSetModelParams. Most of these parameters match the parameters described in section Runmodel.

Parameter	Description
User	The user on whose behalf the Master Template is run. For example, the value for this parameter is passed to the ListModels and the CheckModelAccess exit point scripts.

Parameter	Description
Master Template	The Master Template to be run.
Label	The label of the Master Template run. The value is shown on the HTML forms of the Master Template run.
Keys	A semicolon separated list of keys to be used by the Master Template run. It is possible to refer to files in the session by prefixing their name with "session." For example, key1;session:data;key3;key4.
Extras	A semicolon separated list of extras to be used by the Master Template run.
ResultFileName	The file name of the result document. The result document is always stored as a session file.
ProcessPreviewParams	The value of this parameter is passed to the ProcessPreview exit point script.
ProcessResultParams	The value of this parameter is passed to the ProcessResult exit point script.
ProcessConfirmParams	The value of this parameter is passed to the ProcessParams parameter of the exitpoint scripts.
Environment	The environment under which the Master Template run takes place.
ResultFileFormat	The format of the result document, either "PDF" or "native".
Preview	The HTML forms of the Master Template run are accompanied with previews of the document as it is at that moment. Either "Y" or "N".
PreviewFileFormat	The format of the preview document. Either "PDF", "native" or "HTML".
Confirm	A confirmation page is presented after the model completes. Either "Y" or "N".
ConfirmFileFormat	The format of the confirm document. Either "PDF", "native" or "HTML".
DatabaseUID	User Id to be passed to the DataManager on KCM Core.
DatabasePWD	Password to be passed to the DataManager on KCM Core.
OnSuccessCommand	OnSuccess command, executed when the model succeeds (AS/400 only).
OnFailureCommand	OnFailure command, executed when the model fails (AS/400 only)
PreCommand	Pre command, executed at the start of the run (AS/400 only).
PostCommand	Post command, executed at the end of the run (AS/400 only).

Parameter	Description
DBB_XMLInput	The location of a data XML file that is used to fill the Data Backbone of the Master Template. That data XML file must match the xsd of the Data Backbone. It is possible to refer to files in the session by prefixing their name with "session." For example, "session:xml-input.xml". Uploading a file to the session can be done with the ITPOLSSessionUploadFile service.
DBB_XMLOutput	The location where the XML file with the data of the Data Backbone of the Master Template is stored after the Master Template run completed. It is possible to refer to files in the session by prefixing their name with "session." For example, "session:xml-output.xml". Downloading the file from the session can be done with the ITPOLSSessionDownloadFile service.
RedirectUrl	An URL that can be retrieved with the ITPOLSSessionGetRedirectURL service using an exchange data. Typically used to redirect the user's browser after completing the Master Template.

For certain parameters it is possible to indicate that the value is determined by the configuration of KCM ComposerUI, instead of by the value passed to the service ITPOLSSessionSetModelParams. This can be indicated by passing the special value "*DEFAULT". This special value is supported for the following parameters:

- User
- Environment
- ResultFileFormat
- Preview
- DatabaseUID
- DatabasePWD
- OnSuccessCommand
- OnFailureCommand
- PreCommand
- PostCommand

For the remaining parameters, the Master Template run always uses the exact values specified as parameters to this service.

ITPOLSSessionPrepareModelList

Parameters

```
Parameter Text User
Parameter Text Pattern = ""
Parameter Text Root = ""
Parameter Text Keys = ""
Parameter Text Extras = ""
Parameter Text ResultFileName = "Result"
Parameter Text ProcessPreviewParams = ""
Parameter Text ProcessResultParams = ""
Parameter Text ProcessConfirmParams = ""
```

```
Parameter Text Environment = ""
Parameter Text ResultFileFormat = ""
Parameter Text Preview = ""
Parameter Text PreviewFileFormat = ""
Parameter Text Confirm = ""
Parameter Text ConfirmFileFormat = ""
Parameter Text DatabaseUID = ""
Parameter Text DatabasePWD = ""
Parameter Text OnSuccessCommand = ""
Parameter Text OnFailureCommand = ""
Parameter Text PreCommand = ""
Parameter Text PostCommand = ""
Parameter Boolean UploadDataFile = False
Parameter Boolean UploadDBB_XML = False
Parameter Text DBB_XMLInput = ""
Parameter Text DBB_XMLOutput = ""
Parameter Text RedirectUrl = ""
```

ITPOLSSessionPrepareModelList combines the functionality of the ITPOLSSessionStart, ITPOLSSessionUploadFile and ITPOLSSessionSetModelListParams services. It allows for the definition of KCM ComposerUI Master Template run in a single call and does the following:

- Call `ITPOLSSessionStart`
- If **UploadDataFile** is set to `True`, do a `ReceiveFile` with the value "datafile" for the parameter `Src`. The location of the received data file is used as **Keys**, and the **Keys** value passed to `ITPOLSSessionPrepareModelList` is ignored.
- If **UploadDBB_XML** is set to `True`, do a `ReceiveFile` with the value "xml-input" for the parameter `Src`. The received data file is stored in the session and a reference to that file ("session:xml-input.xml") is used as **DBB_XMLInput** value, and the **DBB_XMLInput** value passed to `ITPOLSSessionPrepareModel` is ignored.
- Call **ITPOLSSessionSetModelListParams** with the passed parameters.

The UploadDataFile feature is designed for Master Templates that use an XML File connection. The uploaded file is automatically used as the XML data file by the Master Template run.

The UploadDBB_XML feature is designed for Master Templates that have a Data Backbone which should be filled by a data XML that matches the xsd of the Data Backbone.

See ITPOLSSessionSetModelListParams for detailed parameter descriptions.

ITPOLSSessionPrepareLetterbook

Parameters

```
Parameter Text User
Parameter Text Letterbook = ""
Parameter Text Keys = ""
Parameter Text Extras = ""
Parameter Text ResultFileName = "Result"
Parameter Text ProcessPreviewParams = ""
Parameter Text ProcessResultParams = ""
Parameter Text ProcessConfirmParams = ""
Parameter Text Environment = ""
Parameter Text ResultFileFormat = ""
Parameter Text Preview = ""
Parameter Text PreviewFileFormat = ""
Parameter Text Confirm = ""
Parameter Text ConfirmFileFormat = ""
```

```
Parameter Text DatabaseUID = ""
Parameter Text DatabasePWD = ""
Parameter Text OnSuccessCommand = ""
Parameter Text OnFailureCommand = ""
Parameter Text PreCommand = ""
Parameter Text PostCommand = ""
Parameter Boolean UploadDataFile = False
Parameter Boolean UploadDBB_XML = False
Parameter Text DBB_XMLInput = ""
Parameter Text DBB_XMLOutput = ""
Parameter Text RedirectUrl = ""
```

The parameters defining a Master Template list and the parameters for a subsequent Master Template run may be set through ITPOLSSessionSetListParams. Most of these parameters match the parameters described in the section Runmodel.

Parameter	Description
User	The user on whose behalf the Master Template list and the subsequent Master Template are run. For example, the value for this parameter is passed to the ListModels and the CheckModelAccess exit point scripts.
Letterbook	The entry point of a Letterbook. This selects the folders and Templates that are available for the browsing user. This is the name of an interactive Letterbook, optionally followed by sub folders separated by slashes (/). Starting with KCM Core and KCM Repository version 4.2.3y when a `rep:/ uri` is provided for the Letterbook, the Letterbook is retrieved from the KCM Repository. See also RetrieveRepositoryObject.
Keys	A semicolon separated list of keys is used by the Master Template run. It is possible to refer to files in the session by prefixing their name with "session." For example, key1;session:data;key3;key4.
Extras	A semicolon separated list of extras is used by the Master Template run.
ResultFileName	The file name of the result document. The result document is always stored as a session file.
ProcessPreviewParams	The value of this parameter is passed to the ProcessPreview exit point script.
ProcessResultParams	The value of this parameter is passed to the ProcessResult exit point script.
ProcessConfirmParams	The value of this parameter is passed to the ProcessParams parameter of the exit point scripts.
Environment	The environment under which the Master Template run will take place.
ResultFileFormat	The format of the result document, either "PDF" or "native".
Preview	Whether or not the HTML forms of the model run should be accompanied with previews of the document as it is at that moment. Either "Y" or "N".

Parameter	Description
PreviewFileFormat	The format of the preview document, either "PDF", "native" or "HTML".
Confirm	A confirmation page is presented after the Master Template completes. Either "Y" or "N".
ConfirmFileFormat	The format of the confirm document, either "PDF", "native" or "HTML".
DatabaseUID	User Id is passed to the DataManager on KCM Core.
DatabasePWD	Password is passed to the DataManager on KCM Core.
OnSuccessCommand	OnSuccess command, executed when the model succeeds (AS/400 only).
OnFailureCommand	OnFailure command, executed when the model fails (AS/400 only).
PreCommand	Pre command, executed at the start of the run (AS/400 only).
PostCommand	Post command, executed at the end of the run (AS/400 only).
UploadDataFile	Either True or False. If UploadDataFile is set to True, a ReceiveFile is performed with the value "datafile" for the parameter Src. In this case the location of the received data file is used as Keys, replacing the value passed to this service. The UploadDataFile feature is designed for use by Master Templates that use an XML File connection (DID). The uploaded file is automatically used as the XML data file by the Master Template run.
UploadDBB_XML	Either True or False. If UploadDBB_XML is set to True, a ReceiveFile is performed with the value "xml-input" for the parameter Src. The received data file is stored in the session and a reference to that file ("session:xml-input.xml") is used as DBB_XMLInput value, replacing the value passed to this service. The UploadDBB_XML feature is designed for use by Master Templates that have a Data Backbone which are filled by a data XML that matches the xsd of the Data Backbone.
DBB_XMLInput	The location of a data XML file is used to fill the Data Backbone of the Master Template. That data XML file must match the xsd of the Data Backbone. It is possible to refer to files in the session by prefixing their name with "session." For example, "session:xml-input.xml". Uploading a file to the session can be done with the ITPOLSSessionUploadFile service.
DBB_XMLOutput	The location where the XML file with the data of the Data Backbone of the Master Template should be stored after the Master Template run completed. It is possible to refer to files in the session by prefixing their name with "session." For example, "session:xml-output.xml". Downloading the file from the session can be done with the ITPOLSSessionDownloadFile service.

Parameter	Description
RedirectUrl	A URL that can be retrieved with the ITPOLSSessionGetRedirectURL service using an exchange data. Typically used to redirect the user's browser after completing the model.

For certain parameters it is possible to indicate that the value should be determined by the configuration of KCM ComposerUI, instead of by the value passed to the service ITPOLSSessionPrepareLetterbook. This can be indicated by passing the special value "*DEFAULT". This special value is supported by the following parameters:

- User
- Letterbook
- Environment
- ResultFileFormat
- Preview
- DatabaseUID
- DatabasePWD
- OnSuccessCommand
- OnFailureCommand
- PreCommand
- PostCommand

For the remaining parameters, the Master Template run always use the exact values specified as parameters to this service.

Exit points

The business application is notified when a Master Template run is completed. KCM ComposerUI offers a number of exit point scripts, because the requirements on how to achieve this may vary widely. See CM Core: OnLine exit points.

It is important to note that these exit point scripts run within Master Template run session. This means that any key/value pairs and files that are stored in the session can be accessed from the scripts.

Chapter 9

KCM Core: ComposerUI exit points

The functionality that supports the production of interactive documents on KCM Core is implemented as a number of KCM Core scripts. Some of these scripts are "exit points," as they can be overridden. These scripts are located in the scripts\User Library subfolder of the KCM Core KCM Work folder.

CheckModelPathAccess.dss (previously CheckModelAccess)

These scripts are called before executing a Master Template. They are used to check whether the given user has access to the given model. If access is not allowed, the scripts display an error and Master Template execution is subsequently aborted.

The CheckModelPathAccess.dss script is called for KCM ComposerUI requests.

Note The CheckModelPathAccess.dss script installed by KCM Core does not implement any validations and permits all requests.

Parameters

- **Master Template**: The Master Template to be executed.
  For CheckModelPathAccess, this is the unmodified Master Template as passed to KCM ComposerUI.
- **UserID**: The KCM ComposerUI user submitting the request.
- **ApplicationID**: The KCM ComposerUI Application ID.

ErrorOccurred.dss

This script is called after an error occurs during KCM ComposerUI Master Template run. By default, this script does nothing. For example, it can be used to do some post-processing of the XML containing the error data or, to inform calling applications of the error occurred. A modified error XML file must be saved to the same location as the original file.

Parameters

The script is called with the following parameters:

- **ITPCode**: The code indicating the source of the error. For example, "itp.Server" or "itp.authorization".
- **ITPReason**: The error message text.
- **ITPLOG**: Messages from the ITPLOG if any.
- **ITPLOGDM**: Messages from the ITPLOGDM if any.
- **ErrorFile**: The file name of the XML file containing the error data.

MakeHTMLDocument.dss

This script is always called when an HTML document is produced by KCM ComposerUI. By default this script uses the Microsoft XSLT transformation tool with a sample XSLT sheet to generate the HTML. We recommend that users change this script to use a custom XSLT sheet and add additional functionality to the document production if required.

Parameters

The script is called with the following parameters:

- **Document**: The file path of the input document. This is in the AiaDocXML format.
- **HTMLResult**: The name of the resulting HTML document. The script must either produce this file or throw an error.
- **IsFinalResult**: Indicates if this is the final result document (True) or a preview document (False).
- **Environment**: The environment that is used to produce the document.
- **Master Template**: The Master Template that produced the document.
- **UserID**: The user name of the browsing user.
- **ApplicationID**: The name of the KCM ComposerUI application.
- **ProcessParams**: The content of the pvw_proc_params (if IsFinalResult=False) or the res_proc_params (if IsFinalResult=True) that is passed to the runmodel call if any.

MakePDFDocument.dss

This script is always called when a PDF document is produced by KCM ComposerUI. By default this script uses the DocToPDF command to generate the PDF document. Users can change the script to use another PDF converter or add additional functionality to the document production.

Parameters

The script is called with the following parameters:

- **Document**: The file path of the input document.
- **PDFResult**: The file path of the resulting PDF document. The script must either produce this file or throw an error.
- **IsFinalResult**: Indicates if this is the final result document (True) or a preview document (False).
- **Environment**: The environment to produce the document.
- **Master Template**: The Master Template to produce the document.
- **UserID**: The user name of the browsing user.
- **ApplicationID**: The name of the KCM ComposerUI application.
- **ProcessParams**: The content of the pvw_proc_params (if IsFinalResult=False) or the res_proc_params (if IsFinalResult=True) that is passed to the runmodel call if any.

ModelRunCompleted.dss

This script is called after an KCM ComposerUI Master Template is run and the result document is written to its folder. By default, this script does nothing. You can add custom functionality to this script. For example, you can pass the file name of the produced document back to a calling application.

Parameters

The script is called with the following parameters:

- **Document**: The file path of the produced document. This parameter is empty when the document is not stored on the server (res_srv=N on KCM ComposerUI).
- **MetaData**: The file path of the Master Template run XML metadata. For more information on Master Template run XML metadata, see the "Integration" chapter in the *KCM Core Developer's Guide*.
- **Environment**: The environment to produce the document.
- **Master Template**: The Master Template to produce the document.
- **Format**: The requested output format.
- **UserID**: The user name of the browsing user.
- **ApplicationID**: The name of the KCM ComposerUI application.
- **ProcessParams**: The content of the res_proc_params that is passed to the runmodel call if any.

PrepareSuspendSession.dss

This script is called when the user clicks the Suspend button on a form. It is run within the same KCM Core job that processes the Master Template run. All ccm_parameters set by the Master Template run are still available in this exit point. Note, that these are no longer available in the exit point SuspendSession.

Parameters

The script is called with the following parameters:

- **Environment**: The KCM Core environment that is used.
- **Master Template**: The Master Template that produces the document.
- **UserID**: The name of the user for whom the Master Template run is done.
- **ApplicationID**: The name of the used KCM ComposerUI application.

ProcessResult.dss

This script is always called after a result document is produced, but before it is written to its folder. By default this script does nothing. You can add functionality to this script for processing the result before presenting it to the user.

Parameters

The script is called with the following parameters:

- **Document**: The file path of the produced document. If you adapt the script to process this document. We recommend, that you save it to the same file.
- **MetaData**: The file path of the Master Template run XML metadata. For more information on model run XML metadata, see the "Integration" chapter of the *KCM Core Developer's Guide*.
- **Environment**: The environment that is used to produce the document.
- **Master Template**: The Master Template that produces the document.
- **Format**: The requested output format.
- **UserID**: The user name of the browsing user.
- **ApplicationID**: The name of the KCM ComposerUI application.
- **ProcessParams**: The content of the res_proc_params that is passed to the runmodel call if any.

ProcessPreview.dss

This script is always called after a preview document is produced, but before it is written to its folder. By default this script does nothing. Users can add functionality to this script for processing the preview before presenting it to the user.

Parameters

The script is called with the following parameters:

- **Document**: The file path of the produced document. If you adapt the script to process this document, we recommend that you save it to the same file.
- **Environment**: The environment to produce the document.
- **Master Template**: The Master Template to produce the document.
- **Format**: The requested output format.
- **UserID**: The user name of the browsing user
- **ApplicationID**: The name of the KCM ComposerUI application.
- **ProcessParams**: The content of the pvw_proc_params that is passed to the runmodel call if any.

ResumeSession.dss

This script is called after the usage of the modelresume page. It retrieves and restores a suspended session. The default implementation restores the session data inside the passed on archive. Then, the KCM Core sessionID is sent back to the client, namely modelresume page, where the browser is forwarded to the correct form.

Parameters

The script is called with the following parameters:

- **Param**: The custom parameter, which is passed on from the page modelresume in KCM ComposerUI.
- **ApplicationID**: The name of KCM ComposerUI application in which modelresume.aspx is called.

SessionResumed.dss

This script is called after KCM Core resumes a suspended session and sends back to the client the sessionID of the restored session. This script is run in the context of the restored session.

Parameters

The script is called with the following parameters:

- **Environment**: The KCM Core environment that is used.
- **Master Template**: The Master Template that produces the document.
- **UserID**: The name of the user for whom the Master Template run is done.
- **ApplicationID**: The name of the used KCM ComposerUI application.
- **Param**: The custom parameter, which is passed on from the page modelsuspend in KCM ComposerUI.

SuspendSession.dss

This script is called after the user clicks the Suspend button on an OnLine form. The default implementation saves the KCM session into an archive and sends it to the client, namely modelsuspend page, where it is offered to the user as a download.

Parameters

The script is called with the following parameters:

- **Environment**: The KCM Core environment that is used.
- **Master Template**: The Master Template that produces the document.
- **UserID**: The name of the user for whom the Master Template run is done.
- **ApplicationID**: The name of the used KCM ComposerUI application.
- **Param**: The custom parameter, which is passed on from the page modelsuspend in KCM ComposerUI.

ValidateModel.dss

This script is called when a user selects a Master Template from a prepared Master Template list. The purpose of this script is to validate that the Master Template selected by user is a part of the prepared Master Template list. In this case, the script writes the path to the Master Template in a session variable with the following keys:

- Key defined by the parameter KeyModel of the script
- Key defined by human readable representation of the Master Template
- Key defined by the parameter KeyLabel of the script

If the Master Template is not valid, the error appears in the script.

The default behavior implemented in the script is that it validates the parameter ModelRef against the provided XMLFile. If it is valid, it will set some session variables. Other forms of authorization can be implemented by adapting the script.

Parameters

The script is called with the following parameters:

- **ModelRef**: The Master Template reference to be validated.
- **XMLFile**: The xml file containing the list of Master Templates that the user is allowed to run.
- **KeyModel**: The key that can be used to store the Master Template name in the session.
- **KeyLabel**: The key that can be used to store the label in the session.

Uploaded.dss

This script is always called when KCM ComposerUI uploads a file to KCM Core. The default implementation of this script does nothing. You can change this script to implement processing of the uploaded file.

Parameters

The script is called with the following parameters:

- **File**: The file name and path of the uploaded file.
- **Master Template**: The Master Template running when the upload takes place.
- **Environment**: The environment to upload the file.
- **UserID**: The user name of the browsing user.
- **ApplicationID**: The name of the KCM ComposerUI application.

Chapter 10

KCM ComposerUI APIs

KCM ComposerUI API demonstrates the functionality that supports the production of interactive documents on KCM Core. For example, KCM ComposerUI uses this API to access KCM Core from the web server. The KCM ComposerUI ASP.NET implementation uses a .NET version of the KCM ComposerUI API, whereas the J2EE implementation uses a Java implementation. There is also a KCM ComposerUI API using COM.

KCM ComposerUI API is an officially supported API. This means that it can be accessed directly without the use of one of the KCM ComposerUI implementations, such as a client application that accesses KCM Core directly from the workstation. In order to do this, an implementation of the KCM Core API (either Java, .NET or COM) is installed as part of the application. The application can translate the XForms XML retrieved through the API to an appropriate GUI representation.

This chapter describes the .NET, Java and COM implementations of the KCM ComposerUI API.

.NET API

The .NET API is implemented by the class Aia.ITP.OnLine.Model. This class offers properties and methods for listing KCM Master Templates and running an interactive KCM Master Templates. The Java and COM APIs for KCM ComposerUI are separate from the APIs for KCM Core, while the .NET API for KCM ComposerUI is delivered as an integrated part of the .NET API for KCM Core.

See the *KCM Core Developer's Guide* for information on the installation and use of the KCM Core .NET API.

Aia.ITP.OnLine.Model class

The class Aia.ITP.OnLine.Model offers functionality for running an interactive KCM Master Template using KCM ComposerUI, and for using various miscellaneous features of KCM ComposerUI. The following section describes the properties and methods of the class Aia.ITP.OnLine.Model.

Running an interactive CM model

The class Aia.ITP.OnLine.Model makes it possible to run KCM Master Templates that interact with a user. Running such a Master Template always follows the below procedure:

1. Call the method Start to perform an initial run of the Master Template.
2. Every time the model run returns interaction requests, call the method Continue to send responses to the interaction and re-run the Master Template.

3. After the last interaction request, the Master Template run is complete. Optionally, call the method Finish to allow KCM ComposerUI to clean up after the Master Template run.

So, running an interactive KCM Master Template is always done by calling the method Start to start running the KCM Master Template, and one or more calls to the method Continue. The number of calls depends on the number of FORM statements and other interaction requests in the KCM Master Template.

Every time the KCM Master Template comes across a FORM statement or other interaction request, KCM will send out an XForms XML document that contains the Form questions. This XML data will be stored into the file specified by the property InfoFile of the object Aia.ITP.OnLine.Model. The Form questions can be extracted from this XML, for instance, by using an XSLT transformation, and presented to the user. The answers have to be passed back to the KCM ComposerUI through the parameter response of the call Continue.

Note Interaction requests do not result only from FORM statements in the KCM Master Template. There are several other situations in which XML forms are generated by KCM Core. For instance, if the KCM XML File Connection is used and no XML data file using the property DataFile is specified, KCM Core sends out an XML form. In this Form, the user is asked to select the XML data file that will be used to run the KCM Master Template.

Also, for every Continue call the KCM Master Template is actually re-run completely. This could possibly have adverse side effects if the KCM Master Template or one of the parameters of the call Start is changed, for instance, the database. The developer must ensure that such side effects either cannot take place, or cannot influence the outcome.

Methods

The `Aia.ITP.OnLine.Model` class has the following methods:

- Constructor
- Letterbook
- List
- Start
- Continue
- Finish
- Upload

Constructor

The constructor for Aia.ITP.OnLine.Model constructs a model object with values for all of the required properties.

- **Signature**

```
Model (string host, string port, string jobID, string infoFile);
```

- **Parameters**

  The parameters correspond to the properties `Host, Port, JobID` and `InfoFile`, respectively. After the constructor is run, these properties contain the values specified as parameters to the constructor. See the section "Properties" below for information on the meaning of these properties.

Letterbook

Lists the models in a letterbook.

- **Signature**

```
bool Letterbook (string letterbook);
```

- **Parameters**

  The parameter `letterbook` specifies the entry point of a Letterbook.

- **Return value**

  If the function returns `true`, an XML data file with a list of models and model folders is stored in the file specified by the property `InfoFile`.

  If the function returns `false`, then the request has failed, and the file specified by the property `InfoFile` will contain XML error data.

  An example (List.xml) of the XML data file returned by the method List can be found in the subfolder *Apis\Online\Example XML Files* of the KCM Core installation folder.

List

This method is **deprecated**.

The method `List` retrieves a list of all KCM Master Templates that the user, as indicated by the property `UserID`, is allowed to access.

- **Signature**

```
bool List (string pattern);
```

- **Parameters**

  The parameter `pattern` specifies the entry point of a Letterbook.

- **Return value**

  If the function returns `true`, an XML data file with a list of models and model folders is stored in the file specified by the property `InfoFile`.

  If the function returns `false`, then the request has failed, and the file specified by the property `InfoFile` will contain XML error data.

  An example (List.xml) of the XML data file returned by the method `List` can be found in the subfolder *Apis\Online\Example XML Files* of the KCM Core installation folder.

Start

See Running an interactive ITP model for a description of the functionality of the method Start.

- **Signature**

```
ResultCode Start (string model,
                  string resultDocument,
                  string keys,
                  string extras,
                  string preCMD,
                  string postCMD,
                  string onSuccessCMD,
                  string onFailureCMD);
```

- **Parameters**
  1. **Master Template**: Specifies the KCM Master Template to be run.
  2. **resultDocument**: Specifies the path of the result document. See the documentation on the property `ResultDocument`.
  3. **keys**: Tthe parameter `keys` is used to pass information to KCM Master Template. For example, you can use the parameter `keys` to identify the customer for whom you want to create a policy or an invoice. The KCM Master Template can then use the identifying information to retrieve the full customer name and address.

     Keys are passed as a string of values separated by semicolons (;). The sequence must be the same as the order in which they are expected in the Master Template.

     The property `DataFile` is used to specify an XML data file to be passed to the KCM Master Template. And the special value "`*DataURI`" is passed to indicate that this data file should be used.

     It is possible to specify an empty key by following a semicolon with another one (;;). This will ensure that the empty parameter will count in the sequence.
  4. **Extras**: Extra parameters are used to pass information to KCM Master Template. For example, you can use the extra parameters to pass extra information on the user who runs the KCM Master Template, at run time. Extras are typically used to pass information to the model that is not available from the database and that cannot be derived from the database data.

     The parameter `Extras` is specified as a string of values separated by semicolons (;). The sequence must be the same as the order in which they are expected in the model.

     It is possible to specify an empty parameter by following a semicolon with another one (;;). This ensures that the empty parameter counts in the sequence.
  5. **PreCMD**: AS/400 only. The Pre command is executed after the library list is set.

     Note We recommend that the API user is aware that this Pre command is executed when the Master Template needs to get data from the database. If the Master Template is set up in such a way that interact statements follow on this data access, the Pre command is executed the first time the data is accessed and every time the call `Continue` is called.
  6. **OnSuccessCMD**: AS/400 only. The OnSuccess command is executed if the Master Template is completed successfully.
  7. **OnFailureCMD**: AS/400 only. The OnFailure command is executed if the Master Template fails.
  8. **PostCMD**: AS/400 only. The Post command is executed at the end of the run (after `OnSuccess` or `OnFailure`).

     Note We recommend that the API user is aware that this Post command is executed for the call to **Start** and also for each subsequent call to **Continue**. **This means that the Post command is executed for the call Start** and all its **Continue** calls.

- **Return values**

  The following result values are defined for the methods **Start** and **Continue**:

  1. **Model.ResultCode.Done**

     The Master Template run is completed. The final document has been stored to the location set in the property **ResultDocument**. If the property **History** is enabled, a file with all form answers in XML format has been saved to the location set in the property **InfoFile**.

  2. **Model.ResultCode.Interact**

     A file containing an XML form (XForms 1.0) is saved to the location set in the property **InfoFile**. In addition, if the property **PreviewDocument** has been set, a preview document has been written to the location set in the property **PreviewDocument**.

     The keyselection.xml and interact.xml examples of XML data files, which are returned by the method **Start** in case of key selection or a form, can be found in the folder: `APIs\Online\Example` XML Files of the KCM Core installation.

     The application that uses the KCM ComposerUI .NET API responds to this result by gathering answers to the Form questions in the XML form file, and by subsequently calling **Continue**, passing the answers to the Form questions in the parameter **response**. The format of the parameter **response** must also be XForms 1.0 XML.

     An example (response.xml) of an XML data file that should be passed to the method **Continue** in case of an interaction, can be found in the folder `APIs\Online\Example` XML Files of the KCM Core installation.

  3. **Model.ResultCode.Error**

     An error has occurred. An XML document describing the error is stored at the location set in the property **InfoFile**. After an error has occurred, the session information on the KCM Core is not destroyed. A user is still able to go back to a previous form, and resubmit other form data.

     An example (`error.xml`) of an XML data file returned by the method **Start** in case of an error can be found in the folder `APIs\OnLine\Example` XML Files of the KCM Core installation.

Continue

The method Continue is used to run KCM Master Template. See Running an interactive ITP model for a description of the functionality of this method.

- **Signature**

```
ResultCode Continue (string responseFile, string submission);
```

- **Parameters**

  responseFile: The path to an XML file containing form answers. These form answers must correspond to the form that is returned by the preceding call to **Start** or **Continue**.

  submission: The parameter **submission** indicates the button that the user presses to submit the form. The possible values of this parameter are defined by the XML form definition returned by the preceding call to **Start** or **Continue**, in the element **button**.

- **Return value**

  The method **Continue** returns the same values as the method **Start**.

Finish

The method `Finish` signals KCM Core that the interactive KCM Master Template run is complete. It may be called after a call to the methods `Start` or `Continue` returns `ResultCode.Done`. KCM Core responds to this signal by cleaning up session storage. Calling `Finish` after a Master Template run is not mandatory.

- **Signature**

```
bool Finish ();
```

- **Return value**

  The method `Finish` returns `true` if the Master Template completion signal is successfully processed by KCM Core. It returns false if an error occurred during the submission or processing of the completion signal.

Upload

The method `Upload` is used to upload a file to an KCM ComposerUI session in KCM Core.

- **Signature**

```
string Upload (string filename);
```

- **Parameters**

  The parameter `filename` specifies the local file that is to be uploaded.

- **Return value**

  The method Upload returns an identifier for the file of the form "`file:<id>`", where <id> is the ID of a file that is stored in the session data store on KCM Core.

Properties

The `Aia.ITP.OnLine.Model` class has the following properties:

- Host
- Port
- JobID
- SessionID
- UserID
- ApplicationID
- Environment
- DBUserID
- DBPassword
- History
- DataFile
- DataFileOnServer
- InfoFile
- ResultDocument
- PreviewDocument

- ResultDocumentFormat
- PreviewDocumentFormat
- ResultDocumentOnServer
- PreviewDocumentOnServer
- ResultOverwrite
- FileExtension
- ProcessPreviewParams
- ProcessResultParams
- FormVersion

Host

The property `Host` is a string that specifies the host name of the computer running KCM Core. The name can be specified either in (IPv4) Internet Protocol dotted address notation (a.b.c.d) or as a resolvable host name.

Port

The property `Port` is a string that specifies the port number on which the KCM Core is running. The port can be specified either in numerical format or as a service name that is resolved through any available service databases.

JobID

The property `JobID` is a string that is used to identify the job on the KCM Core. It appears in the KCM Core log files in all log lines that describe the job run.

SessionID

The property `SessionID` is a string that specifies the KCM Core session ID associated with the submitted job. KCM Core session IDs serves two purposes:

- **Mutual exclusion**: The KCM Core guarantees that multiple requests for the same session ID are not handled in parallel by multiple Document Processors. Instead, multiple simultaneous requests with the same session ID are queued and processed in a series.
- **Persistent storage across jobs**: KCM Core services may use the session ID to store information across several KCM Core jobs, so that each job can use data stored by earlier jobs.

During KCM ComposerUI runs of interactive KCM Master Templates, the session ID is used for both purposes. If no session ID is provided, KCM ComposerUI automatically assigns a session ID to the Master Template run during the method Start. This session ID is automatically retrieved and stored in the property SessionID.

UserID

The property `UserID` represents the Windows user ID of the user who makes this request. If this property is not set, the API uses the default user ID value of the calling application that is running.

ApplicationID

The property `ApplicationID` is used by KCM ComposerUI to store additional accounting information. Typically, It is not necessary to set this property.

Environment

The property `Environment` specifies the environment that the Master Template runs in. If not set, the default environment is used if configured.

DBUserID / DBPassword

The properties `DBUserID` and `DBPassword` specify credentials to override the database credentials that are configured in the KCM Core environment configuration. The password is always transferred in encrypted form.

History

The boolean property `History` specifies that a list of all form answers is returned together with the result document. This information is stored in an XML format, in the file specified by the property `InfoFile`. The default value for the property `History` is `false`.

DataFile / DataFileOnServer

The property `DataFile` specifies the file path to an XML data file. This file is used in conjunction with the KCM XML File Connection, and specifies the XML file that is used as input for the Master Template run. If the property DataFile is not set and the KCM Master Template requires an XML data file, an XML form is sent by KCM Core, asking the user to upload an XML file.

If the property `DataFileOnServer` is enabled, then the file path is interpreted as a path accessible from KCM Core. If the property `DataFileOnServer` is disabled, then the .NET API uploads the file to KCM Core from the specified local path. The default value for the property `DataFileOnServer` is `false`.

InfoFile

The property `InfoFile` specifies the path to the file in which all XML data is returned. Its contents depend on the specific method that is called, and on the return value of the method. Among the possible contents are:

- An XML document containing information about the Master Template run, including the form answers that are given, are stored in the file indicated by the property `InfoFile`; when a `Start` or `Continue` call returns `Model.ResultCode.Ready`, and the property `History` is enabled.
- An XForms XML document containing a form which is displayed to the user; when a `Start` or `Continue` call returns `Model.ResultCode.Interact`.
- An XML document containing information on the error that occurs; when a `Start` or `Continue` call returns `Model.ResultCode.Error`.

ResultDocument and related properties

The property `ResultDocument` specifies the file path that is used to store the final result document. It contains the last value that is passed through the `resultDocument` argument to the method `Start`. The result document is written when a `Start` or `Continue` call returns `Model.ResultCode.Done`. If the property `ResultDocumentAllowOverwrite` is disabled and the file with the specified path already exists, an error code is returned instead. The default value for the property `ResultDocumentAllowOverwrite` is `true`.

If the property `ResultDocumentOnServer` is enabled, then the file path is interpreted as a path accessible from KCM Core. If the property `ResultDocumentOnServer` is disabled, then the .NET API downloads the file from KCM Core to the specified local path. The default value for the property `ResultDocumentOnServer` is `false`.

The property `ResultDocumentFormat` is a string that specifies the file format in which the result document is generated. This format can be either `"native"` or case insensitive `"PDF"`. The default is `"native"`, which means that the word processor format of the Master Template itself gets generated.

PreviewDocument and related properties

The property `PreviewDocument` specifies the file path that is used to store preview documents. If this property is empty, then no preview documents are generated. If a file path is specified, then KCM Core generates a preview document when the Master Template run is interrupted for user interaction, such as when a `Start` or `Continue` call returns `Model.ResultCode.Interact`. The preview document is generated in the format specified by the property `PreviewDocumentFormat`. It is allowed to use the same path for both the preview document and the final result document.

If the property `PreviewDocumentOnServer` is enabled, then the file path is interpreted as a path accessible from KCM Core. If the property `PreviewDocumentOnServer` is disabled, then the .NET API downloads the file from KCM Core to the specified local path. The default value for the property `PreviewDocumentOnServer` is `false`.

The property `PreviewDocumentFormat` specifies the file format in which the preview document is generated. This format can be either `"native"` or case insensitive `"PDF"`. The default is `"native"`, which means that the word processor format of the Master Template itself gets generated.

ResultDocumentFileType

The property `ResultDocumentFileType` is a read-only string property which specifies the file type of the result document. Possible values are:

- `unknown`: Unable to determine the type of the file.
- `doc`: Microsoft Word document.
- `docx`: Microsoft Word Open XML document.
- `ps`: PostScript file.
- `pdf`: PDF document.
- `sxw`: OpenOffice.org / StarOffice document.
- `odt`: OpenOffice.org 2.2 document.
- `lwp`: WordPro document.

- `wpd`: WordPerfect document.

ProcessPreviewParams

The property `ProcessPreviewParams` specifies a string that is passed to the exit point ProcessPreview in KCM Core. This can be used freely to pass information to the exit point, such as to specify which action it should take.

ProcessResultParams

The property `ProcessResultParams` specifies a string that is passed to the exit point ProcessResult in KCM Core. This can be used freely to pass information to the exit point. For example, to specify which action it should take.

FormVersion

The property `FormVersion` specifies the maximum XML Forms feature level that is supported by the client of the API. The clients can protect themselves against new features in the form XMLs returned by KCM Core. If a FORM statement in an interactive KCM Master Template uses features that are not supported by the client, KCM Core aborts the Master Template run with an error. The default value for the property `FormVersion` is "0", which indicates that only basic XML forms are supported.

Java API

Java API for KCM ComposerUI consists of two classes:

- Model class
- Job class

Installation

All functionality can be found in the package com.aia_itp.itpols.api, which is provided by the Java archive **itpolsapi.jar**. This archive is placed in the Apis\Online subdirectory of the KCM Core installation folder.

Model Class

This section describes the properties and methods of the Model class as well as error handling. The Model class offers properties and methods for running an interactive KCM Template.

Run an interactive CM Template

With the Model class, it is possible to run KCM Template that can be interactive. Running such a Template is split in two methods:

1. "Start" method for the initial call
2. "Continue" method for continuing the Template after an interact is answered

Running an interactive KCM Template is always done by calling the Start method to start running the KCM Template, and one or more Continue calls, depending on the number of INTERACT statements in

the KCM Template. Every time the KCM Template comes across an INTERACT, KCM sends out an XML Form with the INTERACT questions. A reference to this XML data is returned in the file mentioned in the Info property of the Master Template object. The questions of the INTERACT can be extracted from this XML and presented to the user, such as with an XSLT. The answers are passed back to the KCM Template in the response parameter of the Continue call.

Note If the XML Connection is used and no XML data file is passed in the Data property of the Template, KCM Core sends out an XML Form with an INTERACT as well. In this INTERACT the user is asked to select the XML data file that should be used to run the KCM Template.

Also, for every Continue call, the KCM Template is actually rerun completely. This could have possible side effects if the KCM Template or one of the parameters of the Start call updates, such as the database. We recommend, that the developer makes sure that such side effects cannot take place or do not influence the outcome.

Properties

The Model class has the following properties. All properties have getter and setter methods.

Connection settings.

Both properties below must be specified when constructing a model.

Property	Type	Description
Host	String	Required. A string that contains the name of the server running KCM ComposerUI. The name can be specified either in (IPv4) Internet Protocol dotted address notation (a.b.c.d) or as a resolvable host name.
Port	String	Required. A string that contains the port to connect to KCM ComposerUI. The port can be specified either in numerical format or as a resolvable port name (TCP/IP service name).

Identification

Property	Type	Description
JobId	String	Required. A string that contains the Job Identifier for this job.
UserId	String	Required. The user that is doing this request.
ApplicationId	String	Optional. Additional accounting information.

Property	Type	Description
SessionId	String	This identifier is initialized by KCM ComposerUI during a "Start" call. If a single Model instance is used to manage a complete Master Template run, this will automatically ensure that Continue/Cancel/Finish calls pass the right session id. We recommend that the user of this API remembers this value, and sets it before issuing one of these subsequent calls.

Database info

Property	Type	Description
Environment	String	Optional. The environment that the Master Template runs in. If not set, the default environment will be used (if configured).
DBUserId	String	Optional. Database User ID.
DBPassword	String	Optional. Database password for the given DBUser. If set, these credentials are initially passed to the Start and the Continue call. They override any credentials that have been set in the Connection Configuration file. **Note** The password is **not** encrypted.

Processing modes

Property	Type	Description
History	boolean	Optional. If set to True, return a list of all interact answers (in XML format) together with the final document. False by default.

Input and Output files

Property	Type	Description
Info	String	Required. This property contains the path to the file in which all XML data will be returned. The content of the XML file depends on the call, and on the return value of the call.

Property	Type	Description
Result	String	The UNC path where the final document is stored. It contains the value that is passed as a Result argument to the Start method. API retrieves the document from KCM Core and stores it at the UNC path location that is accessible from the machine, and on which the Java API is installed. We recommend that the user ensures that the API is able to write to this location. Also see the property *DocumentOnServer* for alternative use.
Preview	String	Optional. The UNC path where the previews are stored. A preview is an intermediate result document, as it is at the moment of an interact. Also see *Result*. If no preview filename is specified, no previews are generated. It is allowed to use the same UNC path for both the preview and the final document.
Data	String	Optional. The UNC path to an XML data file. Also see *Result*. This XML data can be passed as an input file to KCM Core. Only used with the KCM XML File Connection. If the Data property is not set and the KCM Template needs an XML data file, an XML Form is sent out with an INTERACT. In this INTERACT the user is asked to select the XML data file.
DocumentOverwrite	boolean	Optional. Specifies whether existing result documents may be overwritten. *True by default.*
ResultFormat	String	Optional. The format of the final document. Can be either "NATIVE" or case insensitive "PDF". Default is "NATIVE", which means that the word processor format of the model itself gets generated.
PreviewFormat	String	Optional. The format of the previews. Can be either "NATIVE" (the default) or "PDF" (case insensitive).

Property	Type	Description
ResultOnServer	boolean	If True, the Result is relative to the KCM Core machine, such as a local path on the KCM Core machine. In that case, KCM Core is responsible for storing the result document at that location. The final document is not transferred to the Java API then. If this value is False, the final document is transferred and stored on the machine where the API is installed. *False by default.*
PreviewOnServer	boolean	Optional. See *DocumentOnServer*. *False by default.*
DataOnServer	boolean	Optional. See *DocumentOnServer*. If True, the API does not transmit any XML data file. *False by default.*
FileExtension	String	Read only. Contains the extension of the result document. Possible values are: `unknown`: Unable to determine the type of the file. `doc`: Microsoft Word document. `docx`: Microsoft Word Open XML document. `ps`: PostScript file. pdf: PDF document. `sxw`: OpenOffice.org/StarOffice document. `odt`: OpenOffice.org 2.2 document. `lwp`: WordPro document. `wpd`: WordPerfect document.
ProcessPreviewParams	String	Optional. A string that is passed to the ProcessPreview exit point. This can be used freely to pass information to the exit point. For example, to specify which action it should take.
ProcessResultParams	String	Optional. A string that is passed to the ProcessResult exit point. This can be used freely to pass information to the exit point, such as to specify which action it should take.

Property	Type	Description
FormVersion	String	Optional. The maximum feature level that is supported by the client of the API. Clients can protect themselves against new features in the INTERACT XMLs returned by KCM Core. Instead of sending back an XML with these features, KCM Core returns an error.

Methods

1. **Model**

   The constructor creates a new Model object.

   ```
   public Model (String host,
                         String port)
   ```

   Parameters:

   - host - the location of KCM ComposerUI
   - port - the port that KCM ComposerUI listens to

2. **Letterbook**

   Lists the models in a letterbook.

   ```
   public boolean Letterbook (String letterbook)
               throws Exception
   ```

   Parameters:

   - letterbook - the entry point of a Letterbook

     Returns:

     false if the function failed. In that case the Info file contains XML error data. If the function succeeds it returns true, and the Info file contains a list of Master Templates.

3. **List**

   This method has been **deprecated**.

   Lists the Master Templates that the user, as indicated by the current set of properties, is allowed to access.

   ```
   public boolean List ()
               throws Exception
     public boolean List (String pattern)
               throws Exception
   ```

   Parameters:

   - pattern (optional) - the pattern to be passed to the ListModels exit point script.

     Returns:

     false if the function failed. In that case the Info file contains XML error data. If the function succeeds it returns true, and the Info file contains a list of Master Templates.

4. **Start**

   The RunModel functionality is split in two method calls: Start and Continue. Start is always called first. Depending on the outcome of the previous call, zero or more Continue calls are needed.

```
public int Start (String model,
                  String result,
                  String keys,
                  String extras)
           throws Exception

  public int Start (String model,
                  String result,
                  String keys,
                  String extras,
                  String preCMD,
                  String postCMD,
                  String onSuccessCMD,
                  String onFailureCMD)
           throws Exception
```

   Parameters:

   - Master Template: The Master Template to run, given as a rep:/ URI or a Letterbook URI.
   - Result: The URI of the file of the result. By default, this URI is a location relative to the user of the Java API itself. In that case, the API retrieves the document from KCM Core and stores it at this location. We recommend that the user ensures that the API is able to write at this location. Also, see the DocumentOnServer flag for alternative use.

     The value passed here can be retrieved as the DocumentURI property.
   - Keys: The keys string, in KCM Core format. There is no keys file support. A special "*DataURI" value refers to the XML file as indicated by the DataURI property. For Master Templates that need access to an XML file either this key and DataURI must be specified, or the actual path, relative to IKCM Core.
   - Extras: The extras string, in KCM Core format. There is no extras file support.
   - PreCMD: AS/400 only. The Pre command is executed after the library list is set.

     Note We recommend that the API user is aware that this Pre command is executed when the Master Template needs to get data from the database. If the Master Template is set up in such a way that interact statements follow on this data access, the pre command is executed the first time the data is accessed and every time the Continue call is called.

   - OnSuccessCMD - AS/400 only. The OnSuccess command is executed if the model is completed successfully.
   - OnFailureCMD - AS/400 only. The OnFailure command is executed if the model failed.
   - PostCMD - AS/400 only. The Post command is executed at the end of the run (after OnSuccess or OnFailure).

     Note We recommend that the API user is aware that this Post command is executed after every run of the Master Template. This means that the Post command is executed for the Start call and all of its Continue calls.

   Returns:

   - An enumeration type that indicates what this call returns. See the Types section.

5. **Continue**

   If Start(), or another Continue() call is returned ModelResultInteract, and the user is prompted to provide answers to the interact questions, Continue() can be called to continue Master Template execution.

   ```
   public int Continue (String response,
                        String submission)
                  throws Exception
   ```

   Parameters:

   - Response: Contains a reference to a file with the answers to an KCM Master Template interact that is returned earlier.
   - Submission: The Submission parameter indicates what the user presses in the INTERACT screen. The possible values of this parameter are passed in the XML File that holds the INTERACT questions, which is element button.
   - Finish: This is called after a ModelResultReady result is returned to indicate that the user accepts the Master Template. KCM Core may use this as a clean-up signal. It is not mandatory that users call Finish.

     Calling Finish clears the session id.

     ```
     public boolean Finish ()
                    throws Exception
     ```

6. **Upload**

   This method sends the file indicated by the filename argument to KCM Core. The file is stored under an automatically generated unique name in the session directory on the server. The server-side name is returned as the return value of this call as a "file:<name>" URI. The name is a relative path with respect to the session directory on the server. If the call fails, an empty string is returned and the info file contains additional error data.

   The stored files at least remain present at the server until Finish is called.

   Upload only works as long as a session id is present, such as between calls of Start and Finish.

   ```
   public String Upload (String filename)
                  throws Exception
   ```

   Parameters:

   - filename - name of the file to upload

     Returns:

     Local name on the server, empty string on error.

Types

There is a special ModelResult enumeration type, a set of int constants, that specifies the possible outcomes of a call:

- ModelResultReady (=0): Master Template runs. The final document is returned in the Result file. If the History flag is set, the Infofile contains all interact data in XML format. This does NOT destroy the session information on the server.
- ModelResultInteract (=1): Info contains an XML with an interact Form (XFoms 1.0). The identification of the replay stage must be included in the interact Form itself. In addition, if Preview is set, a partial document returns in the file indicated by the Preview file. We recommend that the API user reacts to this result by calling Continue with the result of this or any previous Form as an argument. Furthermore, the user must ensure that the Session property is maintained, such as host or port.

- ModelResultError (=2): An error has occurred. Info contains error details in XML format. This does NOT destroy the session information on the server. A user is still able to go back to a previous interact, and resubmit other interact data.
- ModelResultDBUnauthorised (=3): KCM Core is unable to log on to the database due to an authorization error for some "domain." For example, environment/did/... This implies that the Java API itself does not have the corresponding authorization information in its cache for this domain. Info contains an XForm that requests credentials. Basically, the format adheres to the interact Form. We recommend that the user of the API reacts by calling Continue with the result of this Form as an argument, similar to an ordinary interact.

Job Class

The `Job class` provides methods for setting the job parameters and for job submission. This section describes the properties and methods of this class as well as error handling.

Properties

Property	Type	Description
Host	String	A string that contains the name of the server running KCM Core. The name can be specified either in (IPv4) Internet Protocol dotted address notation (a.b.c.d) or as a resolvable host name.
Port	String	A string that contains the port to connect to. The port can be specified either in numerical format or as a resolvable port name (TCP/IP service name).
JobID	String	A string that contains the Job Identifier for this job.
ApplicationID	String	Optional. Additional accounting information.
SessionID	String	Session ID. See Model class.
ConfirmDisconnect	Boolean	Requires KCM Core to wait for confirmation after completing a job.
LastError	String	Read-only. The last error message generated.

Methods

setAdvancedCapabilities method

Allows the user of the application to specify callback objects that are involved in communication with the server:

- Receiving files
- Transmitting files
- Exchanging data

```
public void setAdvancedCapabilities(ITPOLSDataSender sender,
                                    ITPOLSDataReceiver receiver)
  public void setAdvancedCapabilities(ITPOLSDataSender sender,
                                    ITPOLSDataReceiver receiver,
                                    ITPOLSExchangeData exchange_data)
```

The callback classes have to implement the appropriate interfaces:

```
public interface ITPOLSDataReceiver {

  /**
   * Method called when the CCM ComposerUI Server server executes a SendFile command.
   * This method should return either null to indicate that it does not want to receive
 the
   * data or an OutputStream object to which the data will be written.
   * @param DataItem The Dest parameter as passed in
   * the SendFile Src(...) Dest(...) command.
   */
  public OutputStream ITPOLSReceiveData(String DataItem);

  /**
   * Method called when receipt of the data has been finished.
   * When this method is called all data has been written to the
   * OutputStream returned by ITPOLSReceiveData. It is typically
   * used to close the OutputStream object.
   * @param DataItem The Dest parameter as passed in
   * the SendFile Src(...) Dest(...) command.
   * @param out The OutputStream object as returned by ITPOLSReceiveData.
   */
  public void ITPOLSReceiveDataFinished(String DataItem, OutputStream out);
}

public interface ITPOLSDataSender {

/**
   * Method called when the CCM ComposerUI Server server executes a ReceiveFile
 command.
   * This method should return either null to indicate that it does not want to send
 the
   * data or an ITPOLSInputStream object from which the data will be read.
   * <p>
   * The InputStream object that is returned must be wrapped in
   * an ITPOLSInputStream object, because CCM ComposerUI Server requires the size of
 the data
   * to be available before actually sending the data.
   *
   * @param DataItem The Src parameter as passed in the ReceiveFile Src(...) command.
   */
  public ITPOLSInputStream ITPOLSSendData(String DataItem);
/**
   * Method call when sending the data has been finished.
   * When this method is called all data has been sent to the CCM ComposerUI Server
 server.
   * It is typically used to close the InputStream object.
   *
   * @param DataItem The Src parameter as passed in the ReceiveFile Src(...) command.
   * @param in The ITPOLSInputStream object as returned by ITPOLSSendData.
   */
  public void ITPOLSSendDataFinished(String DataItem, ITPOLSInputStream in);
}

public interface ITPOLSExchangeData
{
  /**
```

```
   * Method called when the CCM ComposerUI Server server executes a exchange_data(key,
 value)
   * function.
   * This method should return a (optionally empty) string to the server.
   *
   * @param Key The Key parameter as passed in the exchange_data function.
   * @param Value The Data parameter as passed in the exchange_data function.
   */
  public String ITPOLSExchangeData(String Key, String Value);}
```

submit method

The submit functions takes the service name as the first function parameter, followed by a boolean flag that indicates whether job submission is synchronous.

Optional parameters are:

- User: The user submitting the job. This is "Remote" by default.
- byteCoding: The encoding of the request, either RQST_IN_ASCII or RQST_IN_UNICODE, which is the default.

```
public boolean submit (
                    String service,
                    boolean sync)
              throws Exception
  public boolean submit (
                    String service,
                    boolean sync,
                    String user)
              throws Exception
  public boolean submit (
                    String service,
                    boolean sync,
                    int ByteCoding)
              throws Exception
  public boolean submit (
                    String service,
                    boolean synchronous,
                    String user,
                    int byteCoding)
              throws Exception
```

Parameters are set before submit is called.

```
// Clear the parameter list, for subsequent addParamter() calls
  public void clearParameters()
  // Sets all parameters at once
  public void setParameters(String[] parameters)
  // Adds a single parameter to the parameter list
  public void addParameter(String parameter)
```

Form Version

A Form version expresses the complexity level for features in an XForms interact form. KCM Core distinguishes the following values:

0 - Basic features available in KCM Core 3.1.10

1 - Multiselect, order and group features

2 - Text Block selection

3 - Edit box, radio buttons, read-only questions, group toggle and explode.

4 - Time questions, tables, buttons

5 - Editable Text Block Questions

6 - Text Block preview by date

7 - Additional toggle condition operators supported

8 - Toggling possible on more than one condition

9 - Editable Rich Text Block Questions supported

Through the COM API FormVersion property, a client can indicate the maximum Form Version that is supported. KCM Core protects the client from features with a Form Version that exceeds this value by sending an error message rather than an interact form containing these features.

Furthermore, each interact form indicates the highest feature level that it contains as an attribute of the main itp:interact tag. This attribute is called "itp:feature-level." For example, an interact form XML can be found in the COM/API/Example XML Files subdirectory of the KCM Core installation.

The interact.xml file format

Descriptions

The XML elements are described in the following generic format:

```
<namespace:element
 attribute=
 attribute=
 …
>
 <namespace:subelement/>
 <namespace:subelement/>
 …
</namespace:element>
```

The following notation is used to indicate how often subelements occur.

Notation	Description	At least	At most
(…)?	Element is optional	0	1
(…)+	Element occurs at least once	1	-
(…)*	Element can repeat more than once	0	-
(…|…)	Select one element	N/A	N/A

Example

```
<x:test
 req=
 ( opt= )?
>
 <y:something/>
 ( <y:else/> )*
</x:text>
```

Defines an element `x: text` that has a required attribute `req=` and an optional attribute `opt=`. It always contains an subelement `<y:something>` and zero, one or more `<y:else>` elements.

Namespaces

Namespaces are used throughout the XML structures. Their definitions are omitted from the attribute lists in this document.

The following namespaces are used in the generated interact.xml files.

xmlns	URL	Description
itp	http://www.aia-itp.com/3.1/interact	The structures described in this document
xforms	http://www.w3.org/2002/xforms	XForms 1.0
xsd	http://www.w3.org/2001/XMLSchema	XMLSchema datatypes
type	http://example.info/types	Used for internally defined datatypes

Examples

Examples only show the relevant parts of the XForms XML structure. Nested XML elements that do not add relevant information are collapsed and shown as `<subelemet/>`.

Top-level elements

This section describes the main XML elements that are used to build a KCM XForms interact.xml file.

itp:interact element

```
<itp:interact
 itp:type=
 itp:lang=
 itp:gui_lang=
 itp:version=
 itp:feature-level=
      itp:has-errors=
>
 <itp:header/>
 ( <itp:question/> | <itp:group/> | <itp:table/> )+
 ( <itp:button/> )+
</itp:interact>
```

Description

The `<itp:interact>` element is the root element of the interact.xml file. It contains all questions and associated definitions for the form.

Attributes

Attribute	Description
itp:type	This attribute indicates the type of form that is presented. Currently defined values are: • **keyselection**: The form is a key-selection screen. • **query**: The form is based on a FORM statement. • **content-wizard**: The form is based on a WIZARD statement.
itp:lang	The language currently used by KCM to generate dates, numbers and other language-dependent output.
itp:gui_lang	The language currently used by KCM to interact with the user. Both itp:lang and itp:gui_lang are presented in the format "ll#CC", where: • "ll" is the ISO-639 language code. • "CC" is the ISO-3166 country code. For example, the KCM language "ENG", English localized for the UK, maps to "en-GB". "NLB", Dutch localized for Belgium, maps to "nl-BE".
itp:version	The version of KCM that generates the interact.xml file.
itp:feature-level	Indicates the FORM features that actually occur in the form. Currently defined levels are: 0 - All features introduced before KCM ComposerUI Server 1 - `MULTISELECT, ORDER, BEGINGROUP/ ENDGROUP` 2 - `VIEW` 3 - `EXPANDABLE, EDITBOX, RADIOBUTTONS, READONLY, TOGGLE` 4 - `BEGINTABLE/ ENDTABLE, TIME, RECORSET, SHOW/ SHOWNOT` 5 - `EDITABLE_TEXTBLOCK` questions 6 - Support for Text Block preview by date 7 - CONTAINS, IN, >, >=, <, <= allowed as toggle operator 8 - Toggling allowed on more than one condition 9 - Editable Rich Text Blocks Each level includes the features of the previous levels.

Attribute	Description
itp:has-errors	This attribute has the value "true" to indicate that the form is a re-send of a previously generated and submitted form, including feedback to the user about the errors in the input. In all other cases, the value is "false".

Content

The <itp:interact> element contains an <itp:header> element which defines the title of the form and the header attributes for the XForms definition.

In addition to this header, it contains one <itp:question> element for the QUESTION / TEXTBLOCK and one <itp:button> element for each button on the FORM. Note that a form always contains at least one question and one button.

Example

```
<itp:interact itp:type="query" itp:lang="nl-NL" itp:gui-lang="en-GB" itp:version="3.2.2"
 itp:feature-level="0" xmlns:itp="http://www.aia-itp.com/3.1/interact" xmlns:xforms="http://
www.w3.org/2002/xforms"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:type="http://example.info/types">
<itp:header/>
<itp:question/>
<itp:question/>
<itp:button/>
</itp:interact>
```

itp:header element

```
<itp:header>
 <itp:title/>
 <xforms:model/>
</itp:header>
```

Description

The <itp:header> element defines global properties of the interact.

Attributes

None.

Content

The <itp:title> element specifies the title of the interact.

The <xforms:model> element defines the XForms model for this FORM. See the XForms specification for details on this element.

Example

```
<itp:header>
<itp:title>Customer Mailing</itp:title>
```

```
<xforms:model>
<xsd:schema/>
<xforms:instance/>
<xforms:bind/>
<xforms:submission/>
</xforms:model>
</itp:header>
```

itp:question element

```
<itp:question
( id= )?
>
 ( <itp:order-response/> )?
 ( <itp:keylist-prompt/> )?
 ( <itp:paragraph-set/>
   <itp:textblockserver>
  <itp:server/>
  <itp:port/>
  <itp:environment/>
   </itp:textblockserver> ) ?
 <xforms:.../>
 ( <itp:helptext/> )?
 ( <itp:feedback/> )?
 ( <itp:layout-hint/> )?
</itp:question>
```

Description

The `<itp:question>` element defines a single question on the form. This element contains the XForms representation of the question and some optional components which affect how the question is displayed.

Attributes

Attribute	Description
id	This optional attribute specifies the ID(…) keyword used to identify this question for use with the SHOW/ SHOWNOT keywords.

Content

Element	Description
`<itp:order-response>`	This optional element is used in textblock selection and multiselect questions to indicate that the form should allow the user to order the responses.
`<itp:keylist-prompt>`	This optional element is used in keyselection forms to provide a description for the selection list.
`<itp:interact>`	This element has an `itp:type="keyselection"` attribute.
`<itp:paragraph-set>`	This optional element is used in a textblock selection and specifies the textblock view from which the textblocks can be selected by the user.

Element	Description
<itp:textblockserver>	This optional element is used in a textblock selection and specifies the textblock server and environment from which the textblocks can be selected by the user.
<itp:helptext>	This optional element specifies a helptext for the question. It is only generated if the question has an HELPTEXT element.
<itp:feedback>	This optional element specifies feedback messages for the question. It is only generated if the form was previously rejected because of an invalid response or if one or more ERRORCONDITION statement in the form were triggered.
<itp:layout-hint>	This optional element specifies a layout hint for the question. It is only generated if the question has an LAYOUT element.

The following XForms elements can currently occur within an <itp:question> element.

Element	Used for
<xforms:select>	Any multiselect question with a picklist.
<xforms:select1>	BOOL question, any question with a picklist.
<xforms:input>	NUMBER and TEXT questions.
<xforms:upload>	Text question with a FILE modifier.
(<xforms:submit>)*	Keys in a keyselection form (one <xforms:submit> element for each key displayed).

The list can be expanded in future versions of KCM.

Example

```
<itp:question>
<xforms:input/ >
<itp:helptext>Provide the age of the customer.</itp:helptext>
<itp:feedback>The age should be between 18 and 35.</itp:feedback>
</itp:question>
```

itp:question element (fixed text)

```
<itp:question>
 <itp:text/>
 ( <itp:layout-hint/> )?
</itp:question>
```

Description

This special version of the <itp:question> element is used to present fixed text on the form.

Attributes

None.

Content

Element	Description
<itp:text>	The text.
<itp:layout-hint>	This optional element specifies a layout hint for the question. It is only generated if the question has an LAYOUT element.

Example

```
<itp:question>
<itp:text>This is fixed text</itp:text>
</itp:question>
```

itp:group element

```
<itp:group
 id=
 level=
 expandable=
 expanded=
 toggle-source=
 toggle-value=
 toggle-condition=
>
 <itp:group-label/>
 ( <itp:layout-hint/> )?
 ( <itp:question/> | <itp:group/> | <itp:table/> )+
</itp:group>
```

Description

The <itp:group> element defines a set of <itp:question> elements that are grouped on the form. Interactive clients should provide a visual hint of such grouping.

If the form is interactively presented to a user, the <itp:group> element can also be used to collapse or expand the groups or to toggle the visibility of the group based on the value of another question on the form.

<itp:group> elements can be nested.

Attributes

Element	Description
id	Sequence number that uniquely identifies this group within the form. The id= attribute is used as the N value to identify the expandedN state in the <response> element.

Element	Description
level	Nesting level of the group.
expandable	Indicates whether or not this group can be expanded. If the value is true, an interactive client should render this group as a collapsible/expandable element.
expanded	Indicates the original state of the group. If the value is true, an interactive client should render this group expanded. This value matches the value of the matching `<expanded>` element in the XForms instance data.
toggle-source	Indicates the question that controls whether or not this group should be toggled as visible/hidden. If this attribute is not present an interactive client should always show this group and ignore other toggle-attributes.
toggle-value	Indicates the value on which the group is shown or hidden.
toggle-condition	Indicates the condition on which the group is shown or hidden. Possible conditions are: • **toggle-condition='='** Show the group only if the current value of the toggle-source question matches the toggle-value. • **toggle-condition='<>'** Show the group only if the current value of the toggle-source question does not match the toggle-value. Note that the list of conditions can be expanded in future versions of KCM.

Content

Element	Description
`<itp:group-label>`	This element defines the heading for the group.
`<itp:question>`	One or more questions.
`<itp:group>`	Nested groups of questions.
`<itp:table>`	Table structure containing one or more questions.
`<itp:layout-hint>`	This optional element specifies a layout hint for the group. It is only generated if the group has a LAYOUT element.

Example

```
<itp:header>
<itp:title>Sample interact with a toggled group</itp:title>
<xforms:model>
<xforms:instance>
<response xmlns="">
<question1 />
<question2 />
```

```
<expanded1 />
</response>
</xforms:instance>
</xforms:model>
</itp:header>

<itp:question>
<xforms:input ref="question1">
<xforms:label>Toggle</xforms:label>
</xforms:input>
</itp:question>

<itp:group id="1" level="1" expandable="false" expanded="false" toggle-
source="question1" toggle-value="42" toggle condition="<>">
<itp:group-label>Group</itp:group-label>
<itp:question>
<xforms:input ref="question2">
<xforms:label>Group question</xforms:label>
</xforms:input>
</itp:question>
</itp:group>
```

In this example, the "Group" group, containing the "Group question," is shown if the value of the "Toggle" question is not equal to 42. Otherwise, the group and its contents are hidden.

itp:table element

```
<itp:table
 id=
 rows=
 columns=

>
 <itp:table-label/>
 ( <itp:layout-hint/> )?
 ( <itp:row/> )*
</itp:table>
```

Description

The element `<itp:table>` defines a group of questions structured in a table grid. One or more of the rows in the table can be extended as a rowset.

Note that the KCM Master Template language currently does not allow cells to span multiple rows and/or columns.

Attributes

Element	Description
id	Label that uniquely identifies this table within the form.
row	Declaration of the number of rows in the table. Rowsets are counted as a single row regardless of the number of rows shown.
columns	Declaration of the number of columns in the table.

Content

Element	Description
<itp:table-label>	This element defines the heading of the table.
<itp:layout-hint>	This optional element specifies a layout hint for the table. It is only generated if the table has an LAYOUT element.
<itp:row>	Definition of a row of cells. The number of <itp:row> elements must match the number declared with the rows= attribute.

Example

```
<itp:table id="table1" rows="3" columns="3">
<itp:table-label>Sample table</itp:table-label>
<itp:row/>
<itp:row/>
<itp:row/>
</itp:table>
```

itp:row element

```
<itp:row
 id=
 columns=
 ( xforms:repeat-nodeset=
   ( xforms:repeat-number= )?
   ( maximum-number= )?
 )?
>
 ( <itp:layout-hint/> )?
 ( <itp:cell>
  ( <itp:question/> )?
   </itp:cell> )+
</itp:row>
```

Description

The <itp:row> element defines a row of cells in the table grid. Each row has its own definition. Rowsets are defined as a single row.

Note that the KCM Master Template language currently does not allow cells to span multiple rows and/or columns.

Questions within a recordset are stored in a repeated element of the response XML. The XForms implementation of uses qualified ref= attributes to the XPath of these questions.

Attributes

Element	Description
id	Label that uniquely identifies this row within the table.

Element	Description
columns	Declaration of the number of columns in the table. The value of this attribute currently must match the value of the columns= attribute of the <itp:table> element.
xforms:repeat-nodeset	Element used to embed repeated elements in the XForms instance data.
xforms:repeat-number	Minimum number of rows that is shown in an interactive client.
maximum-number	Maximum number of rows. Clients should not exceed this limit. (Note that XForms 1.0 does not implement such a limitation.)

Content

Element	Description
<itp:layout-hint>	This optional element specifies a layout hint for the row. It is only generated if the row has an LAYOUT element.
<itp:cell>	Contents of the table cell. Cells can either be empty or contain a single question. The number of <itp:cell> elements must match the number declared with the columns= attribute.

Example

```
<xforms:instance>
<response xmlns="">
<question5 />
<question6 />
<question7 />
<recordset1>
<question9>first record</question9>
<question10 />
<question11 />
</recordset1>
<recordset1>
<question9>second record</question9>
<question10 />
<question11 />
</recordset1>
<ITP-interact-step>1</ITP-interact-step>
<ITP-interact-count>6</ITP-interact-count>
<ITP-interact-ID>
 0bfed7a006ca685abfd59f549fb02491
</ITP-interact-ID>
</response>
</xforms:instance>

<itp:table id="table1" rows="3" columns="4">
<itp:table-label>Sample</itp:table-label>

<itp:row id="row1" columns="4"/>

<itp:row id="row2" columns="4">
<itp:cell>
```

```
<itp:question>
<itp:text>Variables</itp:text>
</itp:question>
</itp:cell>
<itp:cell>
<itp:question>
<xforms:input ref="question5">
<xforms:label>TEXT</xforms:label>
</xforms:input>
</itp:question>
</itp:cell>
<itp:cell>
<itp:question>
<xforms:input ref="question6">
<xforms:label>NUMBER</xforms:label>
</xforms:input>
</itp:question>
</itp:cell>
<itp:cell>
<itp:question>
<xforms:select ref="question7" appearance="full">
<xforms:label>BOOL</xforms:label>
<xforms:item>
<xforms:value>TRUE</xforms:value>
</xforms:item>
</xforms:select>
</itp:question>
</itp:cell>
</itp:row>

<itp:row id="row3" columns="4" xforms:repeat nodeset="recordset1" xforms:repeat
 number="1" maximum-number="4">
<itp:cell>
<itp:question>
<itp:text>Record set example:</itp:text>
</itp:question>
</itp:cell>
<itp:cell>
<itp:question>
<xforms:input ref="/recordset1/question9">
<xforms:label>TEXT</xforms:label>
</xforms:input>
</itp:question>
</itp:cell>
<itp:cell>
<itp:question>
<xforms:input ref="/recordset1/question10">
<xforms:label>NUMBER</xforms:label>
</xforms:input>
</itp:question>
</itp:cell>
<itp:cell>
<itp:question>
<xforms:select ref="/recordset1/question11" appearance="full">
<xforms:label>BOOL</xforms:label>
<xforms:item>
<xforms:value>TRUE</xforms:value>
</xforms:item>
</xforms:select>
</itp:question>
</itp:cell>
</itp:row>
</itp:table>
```

itp:button element

```
<itp:button
 id=
>
 <xforms:submit/>
 ( <itp:layout-hint/> )?
</itp:button>
```

Description

The <itp:button> element specifies a button that appears on the form. These buttons are used to submit the content back to the server.

Attributes

id: Identification of the button.

Content

<itp:button>: This element defines an XForms submit element that is used to submit the form back to the server. This <xforms:submit> element has an submission= attribute whose value is ultimately passed to the COM API as the action parameter.

<itp:layout-hint>: This optional element specifies a layout hint for the row. It is only generated if the row has an LAYOUT element.

The following buttons can be generated.

id=	Action	Description
ok	ok	Submit form and continue.
cancel	cancel	Cancels the Master Template. This button is only available on key selection screens or if the FORM has an ON EXIT statement.
back	back1	Go back one form. This button is not available on the first form of a Master Template.
next	next	Requests the next screen with keys in a key selection form.

Example

```
<itp:button id="ok">
<xforms:submit submission="ok">
<xforms:label>Ok</xforms:label>
</xforms:submit>
</itp:button>
```

Subelements

This section describes the XML elements that occur as attributes/modifiers to a main element. The section *Occurs within* describes a nested structure of each element that occurs.

itp:cell

```
<itp:cell>
 ( <itp:question/> )?
</itp:environment>
```

Description

Defines a cell within a table definition. This element can currently only contain a question.

Occurs within

`<itp:table><itp:row>`

itp:environment

```
<itp:environment>
 environment
</itp:environment>
```

Description

Defines the environment of the KCM/TextBlock Server that should be queried to retrieve Text Blocks for a Text Block selection.

Occurs within

`<itp:question><itp:textblockserver>`

itp:feedback

```
itp:feedback
 reason=
>
 text
</itp:feedback>
```

Description

Specifies feedback messages when a form is represented to the user due to errors in the response or feedback programmed in the KCM Master Template.

Attributes

reason: Identifies the source of the feedback message. Possible values are "error-condition" if the feedback is triggered by an ERRORCONDITION attribute, or validation if the answer failed to satisfy the base requirements of the question, such as acceptable values and data type.

Occurs within

`<itp:question>`

itp:group-label

```
<itp:group-label>
 text
</itp:group-label>
```

Description

Specifies the heading for the grouped set of questions.

Occurs within

`<itp:group>`

itp:helptext

```
<itp:helptext>
 text
</itp:helptext>
```

Description

Specifies the help text for a question.

Occurs within

`<itp:question>`

itp:keylist-prompt

```
<itp:keylist-prompt>

text

</itp:keylist-prompt>
```

Description

Specifies the description for a keylist selection screen.

Occurs within

```
<itp:question>
```

itp:order

```
<itp:order>
 number
</itp:order>
```

Description

Specifies the sequence number of a textblock in a textblock selection list. Every item in the XForms element representing the textblock view is numbered in the order in which it is retrieved from the textblock server. The renderer can use these fields to order the textblocks visually.

Occurs within

```
<itp: question><xforms:select1><xforms:item>
<itp:question><xforms:select><xforms:item>
```

itp:order-response

```
<itp:order-response/>
```

Description

Indicates that responses should be ordered.

Occurs within

```
<itp:question>
```

itp:paragraph-set

```
<itp:paragraph-set>
 text
</itp:paragraph-set>
```

Description

Indicates the paragraph set from which textblocks are selected.

Occurs within

`<itp:question>`

itp:port

```
<itp:port>
 portnumber
</itp:port>
```

Description

Defines the TCP/IP port for the KCM/TextBlock Server that is queried to retrieve Text Blocks for a Text Block selection.

Occurs within

`<itp:question><itp:textblockserver>`

itp:screen-fields

```
<itp:screen-fields>
 text
</itp:screen-fields>
```

Description

Specifies a user-readable representation of a record in the keyselection screen. The renderer should either reprocess this data or display this line in a fixed-width font.

Occurs within

`<itp:question><xforms:submit>`

itp:server

```
<itp:server>
 hostname or IP address
</itp:server>
```

Description

Defines the TCP/IP hostname or IP address for the KCM/TextBlock Server that should be queried to retrieve Text Blocks for a Text Block selection.

Occurs within

`<itp:question><itp:textblockserver>`

itp:table-label

```
<itp:table-label>
 text
</itp:table-label>
```

Description

Specifies the heading for a table of questions.

Occurs within

`<itp:table>`

itp:text

```
<itp:text>
 text
</itp:text>
```

Description

Fixed text.

Occurs within

`<itp:question>`

itp:textblockserver

```
<itp:textblockserver>
 <itp:server/>
 <itp:port/>
 <itp:environment/>
 <itp:repository-project/>?
 <itp:repository-user/>?
</itp:textblockserver>
```

Description

Specifies the location of the KCM/TextBlock Server and the default project in the KCM Repository that is searched for non-qualified objects.

The optional `<itp:repository-user>` element is provided if the model is started from within the KCM Repository.

Occurs within

`<itp:question>`

itp:title

```
<itp:title>
 text
</itp:title>
```

Description

Title of the form.

Occurs within

`<itp:header>`

The <response> element

The <xforms:instance> element describes the result <response> XML structure that is submitted with the answers to the form.

This structure contains entries for all the questions in the form with their default values. Also, it contains additional information that is used by KCM to track the order of the responses and state of elements in the form.

Processing applications always copy this structure and replace the values that are changed by the user. The additional elements with internal values as used by KCM are copied verbatim. The application is aware of and able to handle new values introduced in future versions of KCM.

Structure

```
<response
 xmlns=""
>
 ( <questionN/> )*
 ( <expandedN/> )*
 ( <recordsetN>
  ( <questionX/> )*
   </recordsetN> )*
 <ITP-interact-step/>
 <ITP-interact-count/>
 <ITP-interact-ID/>
</response>
```

Element

Element	Description
<questionN>	This element contains the value of the Nth question. Note that TEXTBLOCK elements on the form are counted as questions but do not have a representation in the <response> element.
<expandedN>	This element contains the state of the Nth group on the form. If its value is TRUE the group is shown expanded. Otherwise the group is shown collapsed. Clients can update this field in the response to remember the state of the groups when the response was submitted. If the form is shown again the states are remembered.
<recordsetN>	This element contains <questionsX> elements for the questions in the Nth recordset.
<ITP-interact-step>	Identifies the occurrence of this specific form during the execution of the model. This element should be copied verbatim from the XForms instance into the response structure.
<ITP-interact-count>	Identifies the form during the execution of the model. This element should be copied verbatim from the XForms instance into the response structure.
<ITP-interact-ID>	Identifies the form. This element should be copied verbatim from the XForms instance into the response structure.

Formatting

The default values and responses are formatted based on the basic KCM type of the variable where the result is stored. Some of the basic types can be modified by KCM keywords, which affects the format where the results are specified.

TEXT

Text variables can contain any text.

```
<question>This is some text</question>
```

NUMBER

Numerical values must always use the US notation. They are allowed to contain a decimal point and must not contain thousands-separators.

```
<question>3.1415</question>
```

BOOL

The text TRUE is mapped to `TRUE`, and any other values are mapped to `FALSE`.

```
<question>TRUE</question>
```

NUMBER / DATE

Dates must be formatted in the XML Scheme `xsd:date` yyyy-mm-dd format.

```
<question>2005-12-31</question>
```

NUMBER / TIME

Times must be formatted in the XML Scheme xsd:time HH:MM:SS format. Note that this format uses a 24-hour clock.

```
<question>16:08:00</question>
```

TEXT / FILE

Since KCM ComposerUI automatically transfers files for questions that use the FILE keyword, it is necessary to store both the name of the remote file as well as the temporary copy of the local file. KCM ComposerUI requires two elements in the response XML file for such questions:

```
<question>File name as stored on the server</question>
<question-local>File name selected by the user </question-local>
```

MULTISELECT / TEXT

Responses to `MULTISELECT` questions are Base-64 encoded to allow the use of spaces, which would otherwise be interpreted as separators by the XForms specification.

Every response value must be Base-64 encoded and appropriately padded. Each encoded value must be prefixed with an "@" that serves as a separator while decoding the responses.

It is allowed to use whitespace and/or linebreaks to split long encodings over multiple lines but such splits are only allowed between encoded quadruplets.

```
<question>@MQ== @Mg== @Mw==</question>
```

which decodes to the response ('1'; '2'; '3').

ITP-interact-ID

The `<ITP-interact-ID>` uniquely distinguishes each `FORM` statement within a single instance of an KCM Master Template.

The `<ITP-interact-ID>` is not suitable to uniquely identify forms:

- Unrelated forms in different KCM Master Templates can have the same `<ITP-interact-ID>` value

- Any changes in an KCM Master Templates, even unrelated to the form, can affect the <ITP-interact-ID> value

Example

```
<xforms:instance>
<response xmlns="">
<question0>1.000</question0>
<question1>2.000</question1>
<question2>3.000</question2>
<question6>4.000</question3>
<question7>O (EXPAND)</question10>
<expanded1>TRUE</expanded1>
<expanded2>TRUE</expanded2>
<expanded3 />
<ITP-interact-step>1</ITP-interact-step>
<ITP-interact-count>3</ITP-interact-count>
<ITP-interact-ID>
bf4e0c358d5e205c97661f93c4ec3b4b
</ITP-interact-ID>
</response>
</xforms:instance>
```

Key selection

The key selection XForms form contains two major parts:

- The <response> structure, which holds a Base-64 encoded representation of the keys which can be selected on this screen.
- For each key an <xforms:submit> element, which is labeled with the human-readable description of the record.

The client application should use the Keys parameter on the COM API instead of manually generating a response on a key selection form.

A client must return the <response> element unmodified to the COM API and pass the submission=attribute as the selected action to indicate the record that has been chosen.

```
<xsd:schema targetNamespace="http://example.info/types">
<xsd:simpleType name="question0">
<xsd:restriction base="string">
<xsd:length value="11" />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0 />
<key0>QUxGS0kAAAAAAAAA=</key0>
<key1>QU5BVFIAAAAAAAAA=</key1>
<key2>QU5UT04AAAAAAAAA=</key2>
<ITP-interact-step>1</ITP-interact-step>
<ITP-interact-count>2</ITP-interact-count>
<ITP-interact-ID>
a1ca34fceb0dec2e67f064d129633ce8
</ITP-interact-ID>
</response>
</xforms:instance>
```

```
<xforms:bind nodeset="question0" id="question0" type="type:question0" />
<xforms:submission id="keylist0" />
<xforms:submission id="keylist1" />
<xforms:submission id="keylist2" />
<xforms:submission id="next" />
<itp:question>
<xforms:input ref="question0">
<xforms:label>
Unique_five_character_code_bas
</xforms:label>
</xforms:input>
</itp:question>
<itp:question>
<itp:keylist-prompt>Select Customer</itp:keylist-prompt>
<xforms:submit submission="keylist0">
<xforms:label>Select</xforms:label>
<itp:screen-fields>
ALFKI Alfreds Futterkiste Maria Anders
</itp:screen-fields>
</xforms:submit>
<xforms:submit submission="keylist1">
<xforms:label>Select</xforms:label>
<itp:screen-fields>
ANATR Ana Trujillo Emparedados y helados Ana Trujillo
</itp:screen-fields>
</xforms:submit>
<xforms:submit submission="keylist2">
<xforms:label>Select</xforms:label>
<itp:screen-fields>
ANTON Antonio Moreno Taquería Antonio Moreno
</itp:screen-fields>
</xforms:submit>
</itp:question>
```

Representation of KCM FORM elements

This section shows what XForms primitives and structures are used to represent questions on KCM forms.

See *XForms 1.0 reference documentation* for details on XForms functionality.

TEXT question

Text questions are directly mapped onto <xforms:input> elements. Default values are directly written into the XForms instance data. The length attribute is encoded in the XSchema declaration.

```
QUESTION "Text question"

LEN (16)

DFT "dft"

ANSWER text_variable
```

```
<xsd:schema targetNamespace="http://example.info/types">
<xsd:simpleType name="question0">
```

```
<xsd:restriction base="string" />
<xsd:length value='16' />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0>dft</question0>
</response>
</xforms:instance>
<xforms:bind nodeset="question0" id="question0" type="type:question0" />
<itp:question>
<xforms:input ref="question0">
<xforms:label>Text question</xforms:label>
</xforms:input>
</itp:question>
```

NUMBER question

Number questions are directly mapped into `<xforms:input>` elements. Default values are directly written into the XForms instance data. The length attribute is encoded in the XSchema declaration.

```
QUESTION "Number questions"

LEN (12 3)

DFT 43

ANSWER number_variable
```

```
<xsd:schema targetNamespace="http://example.info/types">
<xsd:simpleType name="question0">
<xsd:restriction base="decimal">
<xsd:totalDigits value="12" />
<xsd:fractionDigits value="3" />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0>42.000</question0>
</response>
</xforms:instance>
<xforms:bind nodeset="question0" id="question0" type="type:question0" />
<itp:question>
<xforms:input ref="question0">
<xforms:label>Number question</xforms:label>
</xforms:input>
</itp:question>
```

BOOLEAN question

Boolean questions are represented as a multi-select `<xforms:select>` with only a single option to select. Default values are directly written into the XForms instance data.

```
QUESTION "Boolean question"

DFT FALSE
```

```
ANSWER bool_variable
```

```
<xsd:schema targetNamespace="http://example.info/types">
<xsd:simpleType name="question1">
<xsd:restriction base="string" />
</xsd:simpleType>
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0 />
</response>
</xforms:instance>
<xforms:bind nodeset="question1" id="question1" type="type:question1" />
<itp:question>
<xforms:select ref="question0" appearance="full">
<xforms:label>Boolean question</xforms:label>
<xforms:item>
<xforms:value>TRUE</xforms:value>
</xforms:item>
</xforms:select>
</itp:question>
```

FILE attribute

Text questions with a `FILE` attribute ask for a filename. KCM ComposerUI downloads that file from the client into the server. A `FILE` question requires two elements in the `<response>` structure:

- `<questionX>`: The name of the downloaded file.
- `<questionX-local>`: The name as selected on the client.

An application that generates the `<response>` structure needs only to ensure that the `<questionX>` file exists. The `<questionX-local>` element is optional.

```
QUESTION "File question"

FILE

DFT "C:\My Documents\sample.doc"

ANSWER text_variable
```

```
<xsd:schema targetNamespace="http://example.info/types">
<xsd:simpleType name="question0">
<xsd:restriction base="anyURI" />
</xsd:simpleType>
<xsd:simpleType name="question0-local">
<xsd:restriction base="string" />
</xsd:simpleType>
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0 />
<question0-local>
c:\My Documents\sample.doc
</question0-local>
</response>
</xforms:instance>
<xforms:bind nodeset="question0" id="question0" type="type:question0" />
<xforms:bind nodeset="question0-local" id="question0-local" type="type:question0-local" />
```

```
<itp:question>
<xforms:upload ref="question0">
<xforms:label>File question</xforms:label>
<xforms:filename ref="../question0-local" />
</xforms:upload>
</itp:question>
```

DATE attribute

Number questions with a `DATE` attribute ask for a date to be entered. The interactive clients can represent this with a date picker. The resulting date should be written in the XScheme xsd:date YYYY-MM-DD format.

```
QUESTION "Date question"

DATE

DFT 20060307

ANSWER number_variable
```

```
<xsd:schema targetNamespace="http://example.info/types">
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0>2006-03-07</question0>
</response>
</xforms:instance>
<xforms:bind nodeset="question0" id="question0" type="xsd:date" />
<itp:question>
<xforms:input ref="question0">
<xforms:label>Date question</xforms:label>
</xforms:input>
</itp:question>
```

TIME attribute

Number questions with a TIME attribute ask for a time to be entered. The interactive clients can represent this with a time picker. The resulting time should be written in the XScheme xsd:time HH:MM:SS format.

```
QUESTION "Time question"

DATE

DFT 211632

ANSWER number_variable
```

```
<xsd:schema targetNamespace="http://example.info/types">
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0>21:16:32</question1>
</response>
</xforms:instance>
<xforms:bind nodeset="question0" id="question0" type="xsd:time" />
<itp:question>
<xforms:input ref="question0">
```

```
<xforms:label>Time question</xforms:label>
</xforms:input>
</itp:question>
```

MULTISELECT questions

Multiselect questions expect a list of elements as their response. Since spaces are used as separators in the list, the elements are encoded.

Each element is Base-64 encoded and starts with an "@". Any whitespace is ignored.

```
QUESTION "Text list question"

VALUES ( "qqqwww1";

"qqqwww2";

"qqqwww3";

"qqqwww4";

"The quick brown fox jumps over the lazy dog")

DFT "qqqwww1"

ANSWER text_list
```

```
<xsd:schema targetNamespace="http://example.info/types">
<xsd:simpleType name="question0">
<xsd:restriction base="string" />
</xsd:simpleType>
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0>@cXFxd3d3MQ==</question0>
</response>
</xforms:instance>
<xforms:bind nodeset="question0" id="question0" type="type:question0" />
<itp:question>
<xforms:select ref="question0">
<xforms:label>Text list question</xforms:label>
<xforms:item>
<xforms:label>qqqwww1</xforms:label>
<xforms:value>@cXFxd3d3MQ==</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>qqqwww2</xforms:label>
<xforms:value>@cXFxd3d3Mg==</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>qqqwww3</xforms:label>
<xforms:value>@cXFxd3d3Mw==</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>qqqwww4</xforms:label>
<xforms:value>@cXFxd3d3NA==</xforms:value>
</xforms:item>
<xforms:item>
<xforms:label>
```

```
 The Quick Brown Fox Jumps Over The Lazy Dog.
</xforms:label>
<xforms:value>
 @VGhlIFF1aWNrIEJyb3duIEZveCBKdW1wcyBP
dmVyIFRoZSBMYXp5IERvZy4=
</xforms:value>
</xforms:item>
</xforms:select>
</itp:question>
```

READONLY questions

Readonly questions are tagged in the XForms binding. Values in the `<response>` element are ignored on submission.

```
QUESTION "Text question (read-only)"

READONLY TRUE

DFT "dft"

ANSWER text_variable
```

```
<xsd:schema targetNamespace="http://example.info/types">
<xsd:simpleType name="question0">
<xsd:restriction base="string" />
</xsd:simpleType>
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0>dft</question0>
</response>
</xforms:instance>
<xforms:bind nodeset="question0" id="question0" type="type:question0" readonly="true()" />
<itp:question>
<xforms:input ref="question0">
<xforms:label>Text question (read-only)</xforms:label>
</xforms:input>
</itp:question>
```

Text Block selections

The Text Block selection through the `VIEW` question expect a list of elements formatted in the same fashion as the `MULTISELECT` questions do. Only the Text Block IDs should be encoded in the response.

```
<xsd:schema targetNamespace="http://example.info/types">
<xsd:simpleType name="question0">
<xsd:restriction base="string" />
</xsd:simpleType>
</xsd:schema>
<xforms:instance>
<response xmlns="">
<question0>@NTAwMw== @NTAwNA==</question0>
</response>
</xforms:instance>
<itp:question>
<itp:paragraph-set>Claims</itp:paragraph-set>
<itp:textblockserver>
<itp:server>textblock-server</itp:server>
```

```
<itp:port>7777</itp:port>
<itp:environment>Default</itp:environment>
</itp:textblockserver>
<xforms:select ref="question0">
<xforms:label>Select textblocks</xforms:label>
<xforms:item>
<xforms:label>5003 - Send Invoice</xforms:label>
<xforms:value>@NTAwMw==</xforms:value>
<itp:order>0</itp:order>
</xforms:item>
<xforms:item>
<xforms:label>5004 - Provide Estimate</xforms:label>
<xforms:value>@NTAwNA==</xforms:value>
<itp:order>1</itp:order>
</xforms:item>
<xforms:item>
<xforms:label>5005 - Condition breached - reserve rights</xforms:label>
<xforms:value>@NTAwNQ==</xforms:value>
<itp:order>2</itp:order>
</xforms:item>
… … … …
</xforms:select>
</itp:question>
```

Chapter 11

KCM ComposerUI Server customization APIs

KCM ComposerUI provides a number of built-in APIs for use by custom versions of aspx or jsp pages in KCM ComposerUI custom applications. This chapter describes these APIs. See Behavior for more information about custom aspx and jsp pages.

Customization APIs for KCM ComposerUI ASP.NET

The ASP.NET edition of KCM ComposerUI provides a number of APIs to customize aspx pages. These APIs are distinct from those provided in the JBoss edition. All APIs are provided as static members of public class itp. Among other things, these APIs provide for access to KCM Core, validation of session identifiers, the creation of temporary files, and the storage and retrieval of files on a per-session basis.

CreateITPServerJob

The API `itp.CreateITPServerJob` allows a custom aspx page to call an KCM Core Service.

```
Aia.ITP.Server.Job CreateITPServerJob (Page page,
                                       String service,
                                       params string[] parameters)
```

Parameters:

- The parameter page should be passed a reference to the current aspx `Page` object, which is usually `this`.
- The parameter `service` defines which KCM Core service should be called.
- The parameter `parameters` is a variable length list of strings of arguments that should be passed to the KCM Core service.

The returned object is of type `Aia.ITP.Server.Job`, provided by the KCM Core .NET API. See the KCM Core Developer's Guide for more information on this API. The properties of the returned Job object have been initialized to point to the KCM Core instance that KCM ComposerUI has been configured for. Also, the SessionID property has been pre-filled with the session ID provided as a parameter to the current request. When the object is returned, the job is not yet submitted, so it is possible to add event handlers to the object. For instance, you can add event handlers to send and receive files or to exchange data, and to modify the properties of the object. After preparing the events and properties of the Job object, it can be submitted using the method `Submit`.

This job will be submitted to KCM Core as a batch request and is prioritized accordingly.

Example usage of the CreateITPServerJob API:

```
Aia.ITP.Server.Job job = itp.CreateITPServerJob (this,
                                                 "MyService",
```

```
                                                                    "Parameter1",
                                                                    "Parameter2");
job.Submit();
```

This example calls service `MyService` with parameters `Parameter1` and `Parameter2`.

CreateITPOnLineJob

The API itp.CreateITPOnLineJob allows a custom aspx page to call an KCM Core Service.

```
Aia.ITP.Server.Job CreateITPOnLineJob (Page page,
                                       String service,
                                       params string[] parameters)
```

Parameters:

- The parameter `page` should be passed a reference to the current aspx `Page` object, which is usually `this`.
- The parameter `service` defines which KCM Core service should be called.
- The parameter `parameters` is a variable length list of strings of arguments that should be passed to the KCM Core service.

The returned object is of type `Aia.ITP.Server.Job`, provided by the KCM Core .NET API. See the KCM Core Developer's Guide for more information on this API. The properties of the returned Job object are initialized to point to the KCM Core instance that KCM ComposerUI is configured for. Also, the SessionID property is pre-filled with the session ID provided as a parameter to the current request. When the object is returned, the job is not yet submitted, so it is possible to add event handlers to the object. For example, you can add event handlers to send and receive files or to exchange data, and to modify the properties of the object. After preparing the events and properties of the Job object, it can be submitted using the method `Submit`.

This job will is submitted to KCM Core as an online request and prioritized accordingly.

Example usage of the CreateITPServerJob API:

```
Aia.ITP.Server.Job job = itp.CreateITPOnLineJob (this,
                                                 "MyService",
                                                 "Parameter1",
                                                 "Parameter2");
job.Submit();
```

This example calls service `MyService` with parameters `Parameter1` and `Parameter2`.

GetRequestTemporaryFile

The API `itp.GetRequestTemporaryFile` provides the custom aspx page with the path to a temporary file which is deleted at the end of the request.

```
String GetRequestTemporaryFile (String extension)
```

The parameter `extension` defines the extension of the file name that is generated. The return value is a full path to a storage location.

GetSessionStoragePath

The API `itp.GetSessionStoragePath` provides custom aspx pages with a means of storing files that belong to a session, namely, a Master Template run across requests.

```
String GetSessionStoragePath (String sessionid,
                                                    String element,
                                                    String extension)
```

Parameters:

- The parameter `sessionid` contains the KCM Core session ID for which the storage path is retrieved.
- The parameter `element` indicates the name of the storage element.
- The parameter `extension` defines the extension of the file name that is generated.

The return value is a full path to a storage location. This function provides the following stability guarantee: for the same values of `sessionid`, `element` and `extension`, this function always returns the same file name when used in the same ASP.NET session. If the ASP.NET session is different, or if the ASP.NET session ID is the same but the session has expired since the previous time the function was called, the answer may be different. Because of the stability guarantee, this function can be used to provide short term temporary storage that is valid across requests.

Here is an example of how the GetSessionStoragePath API can be used to get the result document of a prepared model run from the modelend.aspx page, and then download it through a separate URL. In the modelend.aspx page6 the result document is downloaded using an KCM Core service to a session storage path, as follows:

```
String sessionid = Request.QueryString["sessionid"];

// Get a storage path that stays the same in the next page
String path = itp.GetSessionStoragePath (sessionid,
                                    "result", "doc");

// Create a Job object to call ITP/Server service GetMyResult
Aia.ITP.Server.Job job = itp.CreateITPServerJob (this,
                                                    "GetMyResult");

// Register a handler for the FileDownload event that tells
// the Job object to download the file to the path we just
// calculated.
job.FileDownload +=
      delegate (String file)
      {
          return path;
      };

// Submit the job to ITP/Server.
job.Submit();
```

The modelend page now provides the user with a link to a different page "downloadresult.aspx?sessionid=<the session id>" to download the document. This page sends the stored document to the web browser:

```
String sessionid = Request.QueryString["sessionid"];
String path = itp.GetSessionStoragePath (sessionid,
                                    "result", "doc");
Response.ContentType = "application/msword";
Response.TransmitFile(path);
```

ServerCallEx

The API `itp.ServerCallEx` provides the ability to call KCM Core services that use the KCM ComposerUI submission protocol, such as ITPOLSSuspendSession.

```
Stream ServerCallEx (Page page,
                                          string service,
                                          string sessionid,
                                          Stream uploadableStream,
                                          string uploadType,
                                          out string downloadedStreamType,
                                          out string newSessionId,
                                          params string[] parameters)
```

Parameters:

- The parameter `page` is the current ASP.NET Page object, usually passed as `this` parameter.
- The parameter `service` indicates the KCM Core service that should be called.
- The parameter `sessionid` indicates the KCM Core session ID that is used. This is usually `Request.QueryString["sessionid"]`, but can be null to indicate that the script should not be run in an KCM Core session.
- The parameter `uploadableStream` can be used to upload a file to the KCM Core service. If no upload is necessary, this parameter should be null.
- The parameter `uploadType` is used to indicate the type of information that is being uploaded. The file is only uploaded to KCM Core if the called service specifies this same string in the Src parameter of the ReceiveFile command that it uses to receive the file.
- The output parameter `downloadedStreamType` indicates the type of information that is contained in a stream that was downloaded, if any. This corresponds to the value of the Dest parameter of the SendFile command that is issued by the KCM Core service to send the file.
- The output parameter `newSessionId` is filled with the new session ID issued by KCM Core, if it did in fact issue a new one. This session ID is only filled if it is communicated by the called KCM Core service using the exchange_data function, with key "sessionid".
- At the end of the parameter list, you can specify a variable list of parameters that are passed to the called KCM Core service.

The return value of the itp.ServerCallEx API is a `Stream` object for the file that was downloaded from KCM Core, if any. The type of data that was downloaded is indicated by the output parameter downloadedStreamType.

The following example is based on the default implementation of the modelsuspend.aspx exit point. It calls the ITPOLSSuspendSession service on KCM Core to suspend the current session, returning a file stream containing the information of the suspended session:

```
String downloadedStreamType;
String newSessionId;
String sessionid = Request.QueryString["sessionid"];

// Call ITP/Server service ITPOLSSuspendSession.
Stream s = itp.ServerCallEx (this,
                             "ITPOLSSuspendSession",
                             sessionid,
                             null,
                             "",
                             out downloadedStreamType,
```

```
                                out newSessionId,
                                "dummy parameter value");

// Make sure that we received a file with name "session".
if (s == null || downloadedStreamType != "session")
{
    throw new Exception("ITPOLSSuspendSession service sent" +
                    " no file or an unexpected file type \"" +
                                    downloadedStreamType +
                                "\", expecting type \"session\"");
}
```

In this example, the parameter value "dummy parameter value" is passed as a parameter to the ITPOLSSuspendSession service. Because the `uploadableStream` parameter is `null`, no file is uploaded to KCM Core in this call.

Session ID validation functions

The API functions validate a session ID and allow custom web pages to verify the validity of the KCM Core session ID. These functions provide for implementation of custom pages in Secure Mode custom applications. See Securing CM ComposerUI for more information on Secure Mode applications.

Ad hoc requests, which use URL parameters to specify their own parameters, are not allowed in Secure Mode applications. Only prepared requests, either prepared Master Template lists or prepared Master Template runs, are allowed. See Integration for more information on prepared requests. The session ID validation functions verify that the query parameter sessionid of a custom web page corresponds to KCM Core session that represents a prepared request of a particular session type.

Note The usage of session ID validation functions is not sufficient to guarantee the security of a KCM ComposerUI web application. See Securing CM ComposerUI for more information on the security guidelines.

1. **ValidateListModelsSessionId function**

   The API `itp.ValidateListModelsSessionId` allows custom ASP.NET web pages to verify whether or not:

   - KCM Core session ID that is passed by the web client corresponds to a valid KCM Core session
   - KCM Core session is configured for a prepared Master Template list

   ```
   void ValidateListModelsSessionId (String sessionid)
   ```

   Parameters:

   - The parameter `sessionid` specifies the session ID that should be validated

   If the passed session ID is invalid, the API `itp.ValidateListModelsSessionId` throws an exception. The session ID is also considered invalid if it corresponds to a session that is not prepared for listmodels functionality.

2. **ValidateRunModelSessionId function**

   The API `itp.ValidateRunModelSessionId` allows custom ASP.NET web pages to verify whether or not:

   - KCM Core session ID that is passed by the web client corresponds to a valid KCM Core session
   - KCM Core session is configured for a prepared Master Template run

   ```
   void ValidateRunModelSessionId (String sessionid)
   ```

   Parameters:

   - The parameter `sessionid` specifies the session ID that should be validated.

   If the passed session ID is invalid, the API itp.ValidateRunModelSessionId throws an exception. The session ID is also considered invalid if it corresponds to a session that is not prepared for runmodel functionality.

3. **ValidateSessionId function**

   The API `itp.ValidateSessionId` is deprecated. Use one of the specific validation functions instead, such as `ValidateRunModelSessionId`.

SetRunModelSession

The API `itp.SetRunModelSession` allows custom ASP.NET web pages to set a KCM Master Template for a session that is prepared for listmodels functionality. This results in a session prepared for runmodel functionality.

```
void SetRunModelSession (Page page,
                         string sessionid,
                         string model,
                         out string newsessionid)
```

Parameters:

- The parameter `page` should be the current ASP.NET Page object, usually the user should pass `this` parameter
- The parameter `sessionid` specifies the current session ID, which is checked for valid listmodels functionality
- The parameter `model` specifies the KCM Master Template to be run
- The parameter `newsessionid` will receive a new session ID, which will identify a session with runmodel functionality

This function validates the given session ID and throws an exception if the session is invalid, or if the session does not support listmodels functionality.

StringResource and RetrieveStringResource

The itp.StringResource and itp.RetrieveStringResource APIs give access to custom string resources stored in the <lang>_custom.msg file of your application. For more information on .msg files and string resources, see Text and JavaScript behavior.

```
void StringResource (Page page,
                     string resourceKey)
String RetrieveStringResource (Page page,
                               string resourceKey)
```

Parameters:

- The parameter `page` should be the current ASP.NET Page object, usually the user should pass `this` parameter
- The parameter `resourceKey` indicates the name of the value that should be retrieved

The `itp.StringResource` API retrieves the string resource value and immediately writes it to the HTTP response. For instance, the result can be output to the web page. The `itp.RetrieveStringResource` API returns the string resource value to the caller.

WriteError

The itp.WriteError API allows custom ASP.NET web pages to write exceptions to the output using the standard KCM ComposerUI error reporting mechanism.

```
void WriteError (Page page,
                 Exception exception)
```

Parameters:

- The parameter `page` should be the current ASP.NET Page object, usually the user should pass `this` parameter
- The parameter `exception` indicates the error that should be reported

This function can be called at any point in an ASP.NET page. It clears the output that is already generated, generates an error page, and finalizes the HTTP response so that no more output is sent to the client. Typically, it is not necessary to call this function, because any exceptions unhandled by ASP.NET pages are handled by KCM ComposerUI. If it is required to report errors during the generation of a web page, this function provides access to the KCM ComposerUI Server error reporting mechanism.

Chapter 12

Securing KCM ComposerUI

By default, KCM ComposerUI is configured to be used as an intranet application. In the default configuration, it is not designed to be used over the Internet. This chapter describes which steps should be taken to expose KCM ComposerUI on the Internet in a secure way. The functionality described in this chapter is currently only available for KCM ComposerUI ASP.NET.

Securing custom applications

If a KCM ComposerUIASP.NET custom application is going to be exposed to the Internet, it must be configured to run in Secure Mode. To secure the application from Internet attackers, KCM ComposerUI applies strict conditions on the communication between the KCM ComposerUI application and web browser. The configuration setting Secure Mode can be enabled from the Customization section of the application configuration page. Changes to this setting only take full effect when the application is deployed. It can be done from the KCM ComposerUI ASP.NET main configuration page.

When an application is running in Secure Mode, the following changes are applied:

- By default, a file in the application folder is not exposed as a web URL, unless it is explicitly configured to be exposed. The configuration file securemode-urls.xml, which can be found in the root folder of the application, defines which files are exposed through web URLs.
- Ad hoc Master Template runs, which use URL parameters to specify the Master Template run parameters, are not allowed. Only prepared Master Template runs can be used.
- Because the KCM Repository uses ad hoc Master Template runs when testing Master Templates in KCM ComposerUI, it is not possible to use a Secure Mode application to test Master Templates from the KCM Repository.
- For the non-customizable web pages exposed by KCM ComposerUI ASP.NET, strict parameter checks are applied, and the web pages cannot be loaded when these parameter checks fail.
- Users are no longer authenticated using Windows Authentication, because this authentication method is not applicable in an Internet situation. Instead, access to the web pages is restricted to authorized users by verifying the session ID provided by the Internet user. However, configuration pages do still require the use of Windows Authentication, so that they cannot be accessed by unauthenticated Internet users.
- Some internal workings of KCM ComposerUI are modified compared to the non-secure mode so that they no longer pass information through URLs.

Note The SecureSample application that is installed with KCM ComposerUI is written so that it can run when Secure Mode is enabled. If you use the application SecureSample as a starting point for creating a new application, always make sure that you are using the most recent version. We recommend not to use the Sample or Sample2 applications as a starting point for creating a new Secure Mode application, because they are not designed for this purpose.

Also, SecureSample is designed in such a way that it can also be used as a stand-alone letterbook. For this purpose, it contains a page preparelist.aspx, which automatically creates a prepared Master Template list. This page merely serves as an example and should therefore never be used in Secure Mode. See the comments in the preparelist.aspx.cs source for more information on this subject.

Exposing web URLs

In the KCM ComposerUI ASP.NET application that is running in Secure Mode, content in the application folder is not exposed to the web by default. To expose a certain file through a web URL, it must be listed in the file securemode-urls.xml, which can be found in the root of the application folder. Changes to this file are applied when the application is deployed. This can be done from the KCM ComposerUI ASP.NET main configuration page.

The following format of the file securemode-urls.xml is supported:

```
<?xml version="1.0" encoding="UTF-8" ?>
<itp:secure-mode-urls xmlns:itp=
   "http://www.aia-itp.com/namespaces/online-secure-mode-urls/1">
  <itp:exposed>
    <itp:pattern pattern="/modelbegin.aspx" />
    <itp:pattern pattern="/css/*.css" />
    <itp:pattern pattern="*.js" />
  </itp:exposed>
</itp:secure-mode-urls>
```

This example contains three `itp:pattern` entries. Each itp:pattern entry specifies a pattern for URLs that are exposed. All URL patterns are specified relative to the application folder.

Three types of patterns are supported: single URLs, sets of URLs in a specific folder, and sets of URLs in any folder.

- A single URL pattern exposes a single URL in the application. A single URL pattern always starts with a slash (/), and specifies the entire path to the URL that should be exposed. For instance, the first entry in the example specifies pattern `/modelbegin.aspx`. Because the path is relative to the application folder, this exposes the URL `/itp/app/<application name>/modelbegin.aspx`. The pattern only exposes the exact URL that is specified, so the example pattern `/modelbegin.aspx` does not expose `/itp/app/<application name>/some/folder/modelbegin.aspx`.
- A pattern may also specify a set of URLs in a specific folder. This type of pattern also starts with a slash (/), and specifies the entire path to the application subfolder in which URLs should be exposed. Instead of specifying a single URL within the folder, a set of URLs is specified by using wildcard characters. For instance, the second `itp:pattern` entry in the example specifies the pattern `/css/*.css`. This exposes all URLs in the application subfolder /css that end with extension .css, but no URLs in any other folders.

  To specify a set of URLs, patterns can use wildcard characters * and ?. The character * matches any sequence of characters. The character ? matches a single character. In addition, URL patterns can use the construct (a|b|c|...|z) to allow for multiple alternatives. For instance, the pattern `/(modelbegin|modelend).aspx` can be used to expose both `/modelbegin.aspx` and `/modelend.aspx`.

- The third type of pattern exposes a set of URLs in any folder of the application. This type of pattern does not start with a slash (/) and does not specify a path. The pattern specifies only the pattern of the exposed URLs, which are then exposed in all application subfolders. This is shown in the third entry `itp:pattern` entry in the example. The specified pattern *.js exposes all files that end with .js, in all folders of the application.

Customizing securemode-urls.xml

The SecureSample applications that are installed with KCM ComposerUI ASP.NET contain an example securemode-urls.xml. This file displays the URLs that must be exposed for the SecureSample application to function properly. We recommend that the developer always modifies securemode-urls.xml to expose only the URLs that are required by the specific application, when building based on the SecureSample custom application. This may involve exposing additional URLs, but we also recommend removal of any URLs that are not required by the custom application. The following list contains URLs that are provided by KCM ComposerUI ASP.NET and that may need to be exposed or unexposed. The column Exposed by default? indicates whether or not a URL is exposed in the default configuration of SecureSample.

URL	Exposed by default?	Description
`/download.aspx` `/opendocument.aspx`	Yes	Required when PDF previews are used.
`/textblockview.aspx` `/viewtextblock.aspx`	Yes	Required when Text Block preview is used.

`/xml2html.aspx`
`/html2xml.aspx`
`/editorpage.aspx`
`/fieldimage.aspx`
	Yes	Required when Editable Text Block Questions are used in dynamic forms, or when TEXTBLOCK questions are used in FORM statements in KCM Master Templates.

`/empty.aspx`	Yes	Required by the sample applications to display an empty frame.
`/upload.aspx`	Yes	Required when FILE questions are used in FORM statements in KCM Master Templates, and when the ActiveX file upload control is enabled.
`/modelbegin.aspx`	Yes	This is a customizable page, the starting point for all Master Templates runs.
`/runmodel.aspx`	Yes	Required for all Master Templates runs.
`/modelend.aspx`	Yes	This is a customizable page, the end point for all Master Templates runs.
`/modelsuspend.aspx`	No	This is a customizable page, which is loaded when the end user uses the button **Suspend**. The default implementation sends the suspended Master Template run information to the web user as a downloadable file.

URL	Exposed by default?	Description
`/modelresume.aspx`	No	This is a customizable page, which can be used to resume a Master Template run that was suspended by the page modelsuspend.aspx. The default implementation allows the user to upload a file containing Master Template run information. This is considered a security risk, so the page is not exposed by default.
`/modelselect.aspx`	Yes	This is a customizable page, the starting point for all Master Template lists.

`/listmodels.aspx`
`/openfolders.aspx`
`/modelselected.aspx`
	Yes	Required for Master Templates lists, used by the default implementation of modelselect.aspx.

`*.js`
`*.png`
`*.gif`
`*.jpg`
`*.htm`
`*.html`
`*.css`
	Yes	KCM ComposerUI ASP.NET ships files with these extensions that should be available through the web. Because files with these extensions are normally public web content, they are exposed by default in the sample applications.

When exposing custom content through securemode-urls.xml, we recommend to be as specific as possible. The reason for this is that KCM ComposerUI ASP.NET may ship files that should not be exposed, and they could be inadvertently exposed when an overly broad pattern is used. For instance, it is not wise to expose the patterns "*.xsl" or "*.xml", because they expose various internal KCM ComposerUI files. However, it is acceptable to expose the more limited "/myfolder/*.xml".

Secure customization

When customizing an application that is intended to run in Secure Mode, it is necessary to conform to a set of security guidelines. In general, it is always safe to apply styling customizations. Any customizations that involve programming require special attention.

The following is a non-exhaustive list of recommendations to keep in mind while customizing Secure Mode applications. Also be aware that following these guideline may not guarantee a secure customization. Careful thought is always required.

- Whenever possible, require a session ID as the only query parameter. Validate this parameter using the correct validation function before use. See Validation functions for more information on these functions.
- Be aware that multiple KCM Core sessions may be sharing the same ASP.NET session state. Therefore, when you store items in the ASP.NET session state, use the session ID as a part of the key, like this: `Session[ Request.QueryString["sessionid"] + "_my_data_item" ]`.
- Perform **extensive checks** on all query string and form parameters that you use.
- Sometimes it is not possible to determine the set of allowed values for a query parameter at design time. In such cases, it is often possible to determine which values are allowed when generating the

calling web page. The user can determine which values are allowed, store the set of allowed values somewhere in the session state, and validate the parameters against that set.

- Pay attention when using the request parameter collections `Request.Params`, `Request.Form` and `Request.Querystring` to retrieve query parameters. In general, instead of using the collection `Params`, it is advisable to specify exactly which source is intended, such as `Request.QueryString` or `Request.Form`. Furthermore, it is very important to ensure that a parameter is always accessed using the same collection. This prevents the situation where the parameter value from one collection is validated, but the value from another collection is actually used. An easy way to ensure success is to read every parameter value only once.
- Always consider what happens if the value of a query parameter does not match one that is expected because of the logic of the application. For instance, if you generate a page that includes a URL "mythings.aspx?thingnumber=5", then note what happens when a malicious user changes this into "mythings.aspx?thingnumber=7".
- **Never** use untrusted user data, such as query string parameters and form data, as file names or parts of file names. This can be exploited by malicious users in numerous unexpected ways. Instead, use the mechanisms provided by KCM ComposerUI ASP.NET for generating secure request-temporary and session-temporary files. For more information, see the descriptions of functions `itp.GetRequestTemporaryFile` and `itp.GetSessionStoragePath` in the chapter Customization APIs for CM ComposerUI ASP.NET. These functions have the added advantage that the files are automatically cleaned up when the request or session expires.
- **Never** include untrusted user data in command lines. Again, this can be exploited in numerous unexpected ways.
- **Never** rely on client side validation logic. JavaScript validation logic is very useful to provide the user with instant feedback, but a malicious user can easily circumvent the validation.

  Furthermore, JavaScript can be disabled in the client browser. Therefore, server side validation logic is always required.
- **Never** pass untrusted user data to KCM Core without applying validation, or without making sure that KCM Core does not use the data in ways that pose a security risk. For instance, KCM Core should not be allowed to use untrusted user data in file paths, in command lines, or even as extra parameters to a Master Template run.
- Whenever a URL is constructed in the code, always make sure that query string parameters are properly escaped. Unescaped query string parameters may lead to unexpected behavior that may be exploited by malicious users.
- Whenever JavaScript code is generated by the code, always make sure that parameters are properly escaped. Unescaped JavaScript parameters may lead to unexpected behavior that may be exploited by malicious users.

CM ComposerUI pages with parameter checks

KCM ComposerUI ASP.NET exposes a number of web pages whose query parameters are explicitly validated before the web page is generated. Some of these web pages are overridable in the application. For these pages, it is not possible to customize the set of parameters of the query. The overridable web pages that are subject to parameter validation are:

- modelsuspend.aspx
- modelend.aspx

Both of these web pages accept only a single query parameter `sessionid`, and nothing else.

Securing CM ComposerUI Server installation

Configuring an application to run in Secure Mode is not enough to ensure security. Several security measures outside of the custom application need to be taken, both on the server that runs KCM ComposerUI and on the KCM Core installation.

Securing CM ComposerUI

Configuring the application to run in Secure Mode does not ensure security. It is not safe to expose the entire KCM ComposerUI ASP.NET web application to the Internet, unless:

- All custom applications on that server are configured and deployed to run in Secure Mode.
- The custom applications are carefully written according to the guidelines in the section Secure customizations, and do not contain any security vulnerabilities.
- The remainder of the server is shielded from the Internet by a firewall, or secured by some other means.
- Web URLs on the server that are outside the KCM ComposerUI virtual directory are protected by the firewall, or secured by some other means.

If the KCM ComposerUI installation contains Secure Mode as well as non-Secure Mode applications, it is still possible to expose the Secure Mode applications to the Internet. In this case, one must place a firewall between the Internet and KCM ComposerUI ASP.NET. The Internet exposes only the URLs that belong to the Secure Mode applications, and no other URLs.

If the custom applications contain ASP.NET (aspx) pages that use ASP.NET web controls, it may be necessary to expose the web URL /itp/WebResource.axd through the firewall, where itp is the name of the virtual directory of KCM ComposerUI. This URL is used by ASP.NET to expose certain dynamically generated content. The default content and sample applications delivered with KCM ComposerUI ASP.NET do not use ASP.NET web controls and therefore do not require this URL to be exposed.

Securing CM Core

The following change might be required on the KCM Core installation. Do not apply this setting on KCM Core versions that support the setting "DisableValidation." This setting is located on the General tab of the Environments page in the KCM Core Administrator. Disabling validation on KCM Core creates a security issue when used in combination with KCM ComposerUI Secure Mode.

Chapter 13

ActiveX deployment on clients

KCM ComposerUI includes an ActiveX control that implements file upload and advanced editing functionality for Internet Explorer on Windows clients. The ActiveX control is used in the sample applications distributed with KCM ComposerUI and also supported for use in custom KCM ComposerUI applications.

Two versions of the ActiveX control are available:

- Version 2 (ITPOLSActiveX2) is only available in a 32-bit version and only works with 32-bit versions of Internet Explorer
- Version 3 (ITPOLSActiveX3) is available in both 32-bit and 64-bit versions

These controls are marked as safe for scripting.

This chapter describes the configuration and deployment of these controls.

Configuring Internet Explorer

To use the KCM ComposerUI ActiveX controls on a client, the following options must be enabled in Internet Explorer:

- Run ActiveX controls and plug-ins
- Script ActiveX controls marked safe for scripting

The options must be enabled for the zone where the KCM ComposerUI server is located. Depending on this zone, the options could already be enabled by default. The options can be applied manually, or centralized through domain policies or the Internet Explorer Administration Kit.

If these options are not enabled, the ActiveX controls are either be blocked completely or the user is prompted interactively for permission every time they are used.

Depending on the deployment method, additional options may have to be enabled to install the ActiveX control before it can be used. These options are described in the appropriate sections.

KCM ComposerUI does not require the following option and we recommend that you keep it disabled:

- Initialize and script ActiveX controls not marked as safe.

Deploying the ActiveX controls

The ActiveX controls can be deployed using the following methods:

- Download on first use

- Centralized deployment
- Manual installation

Download on first use

The KCM ComposerUI installation includes downloadable packages with the ActiveX controls. If the controls are not already installed, Internet Explorer will attempt to download and install them on first use.

The user must be authorized to install the ActiveX. This requires that the following options are enabled in Internet Explorer for the zone in which the KCM ComposerUI Server is located:

- Download signed ActiveX controls

The following option is not required by KCM ComposerUI and we recommend that you keep it disabled:

- Download unsigned ActiveX controls.

Centralized deployment

KCM ComposerUI includes a stand-alone OnLineActiveX installer for the ITPOLSActiveX3 control. This installer can be used for administrative deployment of the ActiveX control, and is available both as an executable and as an MSI package. It can be found in the KCM ComposerUI installation directory. The MSI package is intended for Groups Policy deployment.

When using the MSI, to do a silent installation you need to add a CMDLINE variable: CMDLINE=/s.

Example: Example

```
msiexec -i OnLineActiveX.MSI /quiet /CMDLINE=/s /log &lt;logfile&gt;
```

The example silently installs the ActiveX and logs the results in <logfile>. Note that the ActiveX might not initially be visible in Internet Explorer Manage add-ons.

The OnLineActiveX installer puts the 32-bit version, and on 64-bit systems also the 64-bit version, of the ActiveX control in the Windows system directory and registers it for all users. This installation requires administrative privileges.

Manual installation

It is possible to manually install the ActiveX controls for troubleshooting and testing purposes. During this installation any problems are reported interactively.

To install the ITPOLSActiveX2 control:

- Locate the ITPOLSActiveX2.cab file in the support subdirectory of the KCM ComposerUI web application directory
- Unpack this cabinet file to a temporary directory
- Right-click the ITPOLSActiveX2.inf file and select the Install option from the context menu

To install the ITPOLSActiveX3 control:

- Locate the ITPOLSActiveX3.cab file in the support subdirectory of the KCM ComposerUI web application directory
- Unpack this cabinet file to a temporary directory
- Right-click the ITPOLSActiveX3.inf file and select the Install option from the context menu

Validating the installation

KCM ComposerUI includes test pages to validate whether or not the ActiveX controls are installed correctly.

- For version 2 open the page `http:// ... /support/ITPOLSActiveX2.htm`
- For version 3 open the page `http:// ... /support/ITPOLSActiveX3.htm`

Replace ... with the base URL of the KCM ComposerUI installation.

This URL loads a test page. If this page shows a KCM icon, the ActiveX is loaded correctly and is accessible for the user. If no icon is shown, the ActiveX failed to install or is not permitted to run.

Chapter 14

Troubleshooting

This chapter gives you an information about issues that you may encounter while using the product. Where applicable, suggested workaround are listed.

First use of KCM ComposerUI for ASP.NET is very slow

The first request to KCM ComposerUI (ASP.NET) may take significant time, sometimes even up to several minutes. This is due to the behavior of Microsoft IIS, that uses worker processes to process requests. The worker processes are recycled after a certain time, certain number of requests, or after a certain amount of idle time. This behavior is configured and can be changed in the Microsoft IIS application pool. You can change the Recycle worker process settings in your application pool properties.

Another cause of slow requests (>30 seconds) is often the Certificate Revocation List checking. See http://blogs.msdn.com/b/pfedev/archive/2008/11/26/best-practice-generatepublisherevidence-in-aspnet-config.aspx for more information. To solve this problem you can change a system-wide setting.

Add the following line to your ASPNET.CONFIG or APP.CONFIG file:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <runtime>
        <generatePublisherEvidence enabled="false"/>
    </runtime>
</configuration>
```

Note the ASPNET.CONFIG file is located in the Framework Directory for the version of the Framework you are using. For example:

- For a 64-bit ASP.NET application:
  %SystemRoot%\Microsoft.NET\Framework64\v2.0.50727
- For a 32-bit ASP.NET application:
  %SystemRoot%\Microsoft.NET\Framework\v2.0.50727

Preview documents are loaded in their own application window

If you have configured OnLine Server to show PDF preview documents, the preview may be loaded in Acrobat reader rather than in a browser frame. This is caused by a setting on the client machine. The sequence below shows how to correct the setting:

- Double-click My Computer on the client machine
- On the Tools menu, click Folder Options

- On the File Types tab, in the Registered file types box, click to select the file type that you want to change, such as PDF extension
- Click Advanced
- In the Edit File Type dialog box, make sure the "Browse in same window" check box is checked
- This ensures that documents can be loaded in browser frames

Result document not opened in Word, error mentions OLE container

The modelend page of the Sample2 application uses an ActiveX control to open the result document in a Word instance. To do so, the ActiveX control looks for an existing instance and if none is found it starts Word. The problem is that Word viewed in a browser counts as an instance but cannot be used by the ActiveX control.

The solution is to open Word before anything else and leave it open. It is important that Word is the first program started or it is started before any application that uses Word in any way. When the ActiveX looks for a Word instance it will find this first opened ordinary Word instance to work with.

Result document not visible on desktop

This issue is probably caused by a Word or OpenOffice.org instance that is running in a background process. The result document is opened in this background process and is not visible on the desktop. In this scenario, the user who is trying the run a Word or OpenOffice.org Model is probably the same user as for the KCM Core account. This problem can be avoided by using a dedicated account to run the KCM Core processes.

Default File Upload method only works with Internet Explorer/ Windows clients

In the default KCM ComposerUI configuration, use an ActiveX control to upload files from the clients. This restricts the file upload feature to users with Internet Explorer on the Windows platform. If a different browser should be supported, switch to a generic HTML Forms File Selection control that is supported by most browsers. See the Customization chapter for more information.

File Upload fails

In the default KCM ComposerUI configuration use an ActiveX control to upload files from the client. If the file upload does not function in Internet Explorer, the ActiveX could not be loaded or ActiveX support is disabled.

See ActiveX deployment on clients chapter for more information on the configuration and deployment of the ActiveX control.

Simultaneous sessions per user will fail

A new session is created for each Master Template run in KCM ComposerUI Server. Each session is based on the user running the Master Template. This means that one user can only run one Master Template at the same time.

"String index out of range: -128" error shown in browser

In rare cases, the error "String index out of range: -128" appears in the client's browser when calling KCM ComposerUI Server. Most likely this error is caused by the computer name of the server being too long.

The NetBIOS name of a computer is limited to 15 bytes. If the computer name of a machine has a name longer than 15 characters, the NetBIOS name is truncated. This causes a mismatch between the computer name and NetBIOS name.

KCM ComposerUI Server internally uses both the machine name and the NetBIOS name of the server running KCM ComposerUI. If the name limit of the NetBIOS causes a mismatch to occur, the error "String index out of range: -128" is generated.

As a solution, the machine name must be changed to a shorter name that matches the NetBIOS requirements.

Part of error message is hidden

If an error occurs, it is displayed in the browser. If the browser frame in which the error is shown is too narrow, only part of the error page is visible. Hover on the error to display a tooltip that includes the entire error message.

Sessions lost when running KCM ComposerUI in an IFrame

Privacy settings in web browsers may restrict the use of third-party session cookies. This may cause KCM ComposerUI, when running in an IFrame, to lose its session state between subsequent forms.

A solution is to add the URL of KCM ComposerUI to the sites that are explicitly allowed to use cookies. In Internet Explorer, this option is available under Internet Options > Privacy > Sites.

Alternatively, configure "P3P headers" on the Web Server. For Microsoft IIS, Microsoft Knowledge Base article KB 324013 describes how to do this.