Kofax Communications Manager

ComposerUI for HTML5 JavaScript API Web Developer's Guide

Version: 5.3.0

Date: 2019-05-29

KOFAX

Table of Contents

Preface

This guide describes how you can integrate KCM ComposerUI for HTML5 in a web application using the KCM ComposerUI JavaScript API. The target audience for the guide is web developers of any business application that requires the integration of interactive document composition.

Related documentation

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

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

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

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

Kofax Communications Manager 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.

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.

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

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

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

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

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

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

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

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

This chapter gives understanding of key concepts of the interactive document composition and cross-origin resource sharing.

Interactive document composition

The following components are involved in the interactive document composition:

- KCM Core
- The business application (server side)
- The business application (web browser, client side)
- The KCM ComposerUI JavaScript API loaded in a page of the business application (web browser, client side)

The process of interactive document composition consists of the following steps:

1. The business application (server side) calls KCM Core through its web services interface. The specific call is ComposeDocxInteractiveStart or ComposePdfInteractiveStart, depending on the required output. During this call, it passes a reference to the object (a Template or a Letter Book) in KCM Designer and the data for the interactive document composition run.
2. KCM Core registers a session for the run and returns a URL and a session identifier.
3. The business application passes this information to its client side.
4. The business application calls the KCM ComposerUI JavaScript API and passes the URL, the session identifier, an identification of an HTML element on the page, and a number of JavaScript callback functions.
5. The KCM ComposerUI JavaScript API connects to KCM Core. Based on the retrieved information, it presents all interaction required for composition of the document to the end user. All interaction is presented inside the HTML element that was indicated by the call to the API.
6. When the interactive process is finished, an appropriate callback function is called by the API. This allows the business application to take up the process and notify its server side that the document is composed.
7. The business application server side calls KCM Core through its web services interface. The specific call is ComposeDocxInteractiveGet or ComposePdfInteractiveGet, matching the previous call. During this call, it passes the session identifier. The call returns the composed document.

Cross-origin resource sharing

KCM Core is accessed directly from the browser while the page it is accessed from resides on the business application web server. This is called cross-origin resource sharing (CORS) and is only allowed under certain conditions. For example, in Microsoft Internet Explorer 10 and higher, these conditions are met if both sites have been added to the list of Trusted Sites and the Security Setting Miscellaneous/ Access data sources across domains is set to Enable. If these conditions are not met in the environment of the business application, we strongly recommend that you route the HTTP traffic to KCM Core through a proxy that shares its base URL with the business application. In this case, CORS-related restrictions do not apply.

Chapter 2

API

The current version of the API is CcmComposerUIAPIV2. For CcmComposerUIAPIV1, see Previous versions.

The description of the API is written with the assumption that the URL and the session identifier are available.

Dependencies

You can download the JavaScript API from the following location on a KCM Core installation.

http://<kcm server>:8081/start/home.html

where <kcm server> is the host name of the server.

The JavaScript is delivered with a corresponding CSS file that can be modified.

The API depends on the following components.

- jQuery:
  1.8 or higher, verified up to 1.12
  2.0 or higher, verified up to 2.2.4
  3.0 or higher, verified up to 3.2.1
- jQuery UI:
  1.10 or higher, verified up to 1.12.1
- TinyMCE, including a number of plug-ins for specific KCM Text Block support. A full tree can be downloaded from the start page.

To prevent conflicting versions to be loaded, the following components may be provided by the including web page as well:

- Moment.js
  2.6 or higher, verified up to 2.18.1
- PDf.js
  1.4 or higher, verified up to 1.7.225

The web page that includes the JavaScript API should be identified to the browser as being encoded in UTF-8.

Calls

The current version of the API exposes one main call, which is CcmComposerUIAPIV2.Run.Start. It has the following parameters:

- **starturl**. String. An absolute URL to KCM Core, which is the absolute version of the relative URL that is retrieved through the web services call. This URL is accessed from the browser on the client machine. The base of this URL has to be clear to the specific network environment.
- **sessionid**. String. The session identifier that is retrieved through the web services call.
- **jobid**. String. An identifier of the specific run. Only used to identify the run in logs and during callbacks.
- **elementid**. String. The HTML id of the element on the page that contains the user interaction for the document composition.

  Note This element must have a sufficient width to correctly display user interaction.

- **callbacks**. CcmComposerUIAPIV2.Model.RunCallbacks. An object that exposes a number of callbacks that are called to notify the business application of an event.
- **options**. CcmComposerUIAPIV2.Model.RunOptions. An object containing some additional options.

The call returns true or false. Any subsequent information is passed through the callbacks.

Also, the API exposes CcmComposerUIAPIV2.Version.Get. This returns the version object with the following attributes:

- **version**. String. The software version of the API.
- **build**. String. The build number of the API.

Objects

You can create the following objects through the API:

- CcmComposerUIAPIV2.Model.RunCallbacks
- CcmComposerUIAPIV2.Model.RunOptions

The following objects are passed during callbacks:

- RunInfo
- Result
- Document
- Error

CcmComposerUIAPIV2.Model.RunCallbacks

The CcmComposerUIAPIV1.Model.RunCallbacks call returns a RunCallbacks object that can be passed to the Start call. The call takes a single object as an argument. From this object, the following attributes are registered as a callback:

- **onruncompleted**. Function(RunInfo, Result). This function is called when the interactive document composition process is completed successfully. During the callback, RunInfo and Result objects are passed. The element identified by the elementid parameter on the Start call contains the latest form presented to the end user during the interactive document composition.

  If the call does not contain any form, see RunInfo for the attribute **hasbeeninteractive**.
- **onrunsuspended**. Function(RunInfo). This function is called when the interactive document composition is suspended by the user. During the callback, a RunInfo object is passed. The element identified by the elementid parameter on the Start call contains the last form presented to the end user during interactive document composition.

  If the call does not contain any form, see RunInfo for the attribute **hasbeeninteractive**.
- **onrunfailed**. Function(RunInfo, Error). This function is called when the interactive document composition is completed in an error state. During the callback, RunInfo and Error objects are passed.
- **onerror**. Function(RunInfo, Error). This function is called when an error occurs during the interactive document composition, but the process continues. During the callback, RunInfo and Error objects are passed.

CcmComposerUIAPIV2.Model.RunOptions

The CcmComposerUIAPIV2.Model.RunOptions call returns a RunOptions object that can be passed to the Start call. The call takes a single object as an argument. From this object, the following attributes are registered as options:

- **locale**. Object. The value of the locale.ui attribute of this object is exploited to determine the user interface language to be used during interactive document composition. The following values are supported by default.

Value	User interface language
en	English
de	German
es	Spanish
fr	French
it	Italian
ja	Japanese
nl	Dutch

Value	User interface language
pt-BR	Brazilian Portuguese

Note For KCM ComposerUI for HTML5, the set of languages is extensible, so other values may apply. For more information, see the chapter "Manage language packs" in the *Kofax Communications Manager Getting Started Guide*.

- **tbescripturl**. String. The relative location of the TinyMCE jQuery file used to implement the Text Block support.

  The default value for this setting: tinymce/tiny_mce_src.js
- **tbeformatfunctions**. Array. An array of format function objects to be presented to the end user in the formatting dialog in the Text Block Editor. A format function consists of the name attribute and the parameters attribute.

  Example `[{name: 'Concat', parameters: ['prefix']}, {name: 'AsNumber', parameters: ['nr_of_decimals']}]`.
- **stylemap**. Object. A map of key/value pairs that determines the actual class names used for a number of logical class names. This allows the caller to determine the look and feel of a well-known subset of the UI using an application specific CSS file.

Keys to pass on the stylemap parameter

The following keys can be passed on the stylemap parameter. If a value is available for a key, this value is applied as a class name to the keys listed in the table. Otherwise, a default value is used.

Key	Default value	Description
active	kccmactive	Key used to mark that a certain element is active. For example, Text Blocks in the selection that is currently presented in the right pane.
highlight	kccmhighlight	Key used to highlight elements. For example, the Text Block that the user points to in the right pane.
error	kccmerror	Key used to mark elements that indicate error states.
selected	kccmselected	Key used to mark that certain elements are selected. For example, the selected item when reviewing a Document Pack.
button	kccmbutton	Button used for the Form submission and for editing an editabletextblock question.
submitbutton	kccmsubmitbutton	Button used for the Form submission.
submitbuttonok	kccmsubmitbuttonok	The Next button advancing interaction to the next Form.
submitbuttonback	kccmsubmitbuttonback	The Back button returning interaction to the previous Form.

Key	Default value	Description
submitbuttonupdate	kccmsubmitbuttonupdate	The Update button applying the value of a Field to the current document.
primarybutton	kccmprimarybutton	Button to represent the primary action on a page. Currently assigned to the Next button, the Update button and the Close button in the Document Pack Review pane.
cwbutton	kccmcwbutton	Button to open a selection pane (Text Block or section selection) for a certain location in the document.
cwbuttonsections	kccmcwbuttonsections	Annotation of cwbutton class for section selection.
cwbuttontextblocks	kccmcwbuttontextblocks	Annotation of cwbutton class for Text Block selection.
tooltip	kccmtooltip	Tooltip that is presented if the user points to a Help icon for a question. **Note** In the HTML output, contrary to all other classes, the tooltip is not a child of the kccmroot.
title	kccmtitle	Title presented above each Form.
subtitle	kccmsubtitle	Subtitle presented immediately below the title, if available.
group	kccmgroup	Group divider containing any number of grouped questions or items.
grouphead	kccmgrouphead	Header divider containing the name of the group.
cwdocument	kccmcwdocument	The canvas on which the content of the document is presented.
cwsidebar	kccmcwsidebar	The sidebar shown alongside the canvas.
cwfield	kccmcwfield	A Field inside the Text Blocks that fill the canvas.
cwfieldeditable	kccmcwfieldeditable	Annotation of cwfield for a Field that can be modified by clicking it.
textblockpreview	kccmtextblockpreview	A preview of a Text Block that is shown for editabletextblock, textblocksingleselect, and textblockmultiselect questions.
textblockpreviewfield	kccmtextblockpreviewfield	A Field inside a Text Block preview.

Key	Default value	Description
textblockeditor	kccmtextblockeditor	The Text Block Editor used to present editing facilities for Text Blocks marked as editable.
questionlabel	kccmquestionlabel	Question label.

key	default value	description
questionlabelerror	kccmquestionlabelerror	The label of a question that has an invalid answer.
letterbookfolder	kccmletterbookfolder	List of elements in a folder in the Letter Book tree.
letterbookfoldertitle	kccmletterbookfoldertitle	Folder list element (node) in the Letter Book tree.
letterbooktemplate	kccmletterbooktemplate	Template list element (leave) in the Letter Book tree.
letterbooktemplatelink	kccmletterbooktemplatelink	Template link that allows the user to select a Template.
statictext	kccmstatictext	Static text on a Form. For example, it is used for more extensive explanation for a number of questions.
textquestion	kccmtextquestion	Question control presented to the user to answer a specific type of question.
textareaquestion	kccmtextareaquestion	Question control presented to the user to answer a specific type of question.
numberquestion	kccmnumberquestion	Question control presented to the user to answer a specific type of question.
boolquestion	kccmboolquestion	Question control presented to the user to answer a specific type of question.
datequestion	kccmdatequestion	Question control presented to the user to answer a specific type of question.
timequestion	kccmtimequestion	Question control presented to the user to answer a specific type of question.
filequestion	kccmfilequestion	Question control presented to the user to answer a specific type of question.
singleselectquestion	kccmsingleselectquestion	Question control presented to the user to answer a specific type of question.

key	default value	description
radiosingleselectquestion	kccmradiosingleselectquestion	Question control presented to the user to answer a specific type of question.
multiselectquestion	kccmmultiselectquestion	Question control presented to the user to answer a specific type of question.
textblocksingleselectquestion	kccmtextblocksingleselectquestion	Question control presented to the user to answer a specific type of question.
textblockmultiselectquestion	kccmtextblockmultiselectquestion	Question control presented to the user to answer a specific type of question.
editabletextblockquestion	kccmeditabletextblockquestion	Question control presented to the user to answer a specific type of question.
editablerichtextblockquestion	kccmeditablerichtextblockquestion	Question control presented to the user to answer a specific type of question.
reviewheader	kccmreviewheader	The header shown when reviewing a Document Pack.

RunInfo

A RunInfo object is passed on the callback function to provide information about the run. A RunInfo object contains values for the following attributes:

- **starturl**. String. The starturl passed to the Start call.
- **sessionid**. String. The sessionid passed to the Start call.
- **jobid**. String. The jobid passed to the Start call.
- **elementid**. String. The elementid passed to the Start call.
- **options**. RunOptions. The options object passed to the Start call.

In case of the onruncompleted callback, the following attribute is also available:

- **hasbeeninteractive**. Boolean. A boolean value that indicates whether any interaction was required to compose the document. Based on this value, the business application may determine whether it makes sense to provide a button that leads the end user back to the last Form of the interactive document composition process.

Result

A Result object is passed on the onruncompleted callback. It has two attributes:

- **type**. Document or documentpack, depending on whether the document composition resulted in a single document or a Document Pack.
- **object**. The actual result. Currently only provided if type is document. In this case, the attribute exposes an object of type Document.

Document

A Document object is passed on the onruncompleted callback. The Document is currently not guaranteed to have any attributes. It offers two methods; one is to retrieve the content of the document, and the other is to update it.

The GetContent method accepts one callback parameter:

- **callback**. Function(format, content). This method is called once the content of the document is retrieved from the server. The format parameter contains the format of the result document (.doc, .docx or .pdf). The content parameter includes the actual content (base64 encoded).

The SetContent method accepts the content as its only parameter:

- **content**. Base64String. The content that is uploaded to the server to replace the currently stored content of the document.

Error

When an error occurs, an Error object is passed to the onrunfailed or onerror callback. This object contains values for the following attributes:

- **type**. String. Currently "http" or "ui."
- **message**. String. A message used to display the error to the end user.
- **details**. Object. An object containing more details about the error. The exact contents of the object differs between error types.

For type "http," the error.details object contains the following attributes:

- **url**. String. The URL that is accessed.
- **statuscode**. Number. The http status code that is returned.
- **statustext**. String. The status text (if any) that is returned in the http header.
- **responsetext**. String. The response text (if any) that is returned in the response body.

For type "ui," the error.details object contains the following attributes:

- **context**. String. A textual representation that indicates the context in which the error occurred.
- **exception**. Object. The JavaScript exception that is thrown to indicate the error.

Edit the result document

In many scenarios, editing a Microsoft Word document immediately after it has been composed is a requirement of the business process. The API contains functionality that allow for a seamless integration of editing in the process.

The document object passed during the onruncompleted callback exposes GetContent and SetContent methods that help retrieve and update the content of the result document. A subsequent call to the ComposeDocxInteractiveGet returns the updated document.

One way of offering the user the content for editing is to use an ActiveX control. ActiveX is a technique that provides useful functionality in many contexts. An ActiveX control that supports editing of base64

Word content is available on a Kofax KCM installation and can be downloaded from the start page. The example from Test an example integration of the JavaScript API shows how the ActiveX control can be implemented to use interactive document composition with post-composition editing of the result document.

Test an example integration of the JavaScript API

To test an example integration of the JavaScript API, use the following web page.

http://<kcm server>:8081/start/home.html

The interactive composition of a document can be tested by clicking "Test" in the section "ComposerUI HTML5." This example runs a fixed Template from an example KCM Designer project. Also, you can download it from the start page and use it as a reference for an integration in the business application, as it provides information on how an integration is achieved.

Previous versions

The previous version of CcmComposerUIAPI is supported, which is CcmComposerUIAPIV1. You can use it with the corresponding calls and objects.

CcmComposerUIAPIV1

Calls

The CcmComposerUIAPIV1 API exposes one main call, which is CcmComposerUIAPIV1.Run.Start. It has the following parameters:

- **starturl**. String. An absolute URL to KCM Core, which is the absolute version of the relative URL that was retrieved through the web services call. This URL is accessed from the browser on the client machine. The base of this URL has to be clear to the specific network environment.
- **sessionid**. String. The session identifier retrieved through the web services call.
- **jobid**. String. An identifier of the specific run. Only used to identify the run in logs and on callbacks.
- **elementid**. String. The HTML ID of the element on the page that contains the user interaction for the document composition process.
- **callbacks**. CcmComposerUIAPIV1.Model.RunCallbacks. An object that exposes a number of callbacks that is called to notify the business application of an event.
- **options**. CcmComposerUIAPIV1.Model.RunOptions. An object containing some additional options.

The call returns true or false. Any subsequent information is passed through the callbacks.

Also, the API exposes CcmComposerUIAPIV1.Version.Get. This returns the version object with the following attributes:

- **version**. String. The software version of the API.
- **build**. String. The build number of the API.

Objects

The following objects can be created through the API:

- CcmComposerUIAPIV1.Model.RunCallbacks
- CcmComposerUIAPIV1.Model.RunOptions

The following objects are passed during callbacks:

- RunInfo
- Document
- Error

CcmComposerUIAPIV1.Model.RunCallbacks

The CcmComposerUIAPIV1.Model.RunCallbacks call returns a RunCallbacks object that can be passed to the Start call. The call takes a single object as an argument. From this object, the following attributes are registered as a callback:

- **onruncompleted**. Function(RunInfo, Result). This function is called when the interactive document composition is completed successfully. During the callback, RunInfo and Result objects are passed. The element identified by the elementid parameter on the Start call contains the last form that was presented to the end user during the interactive document composition.

  If the call does not contain any form, see RunInfo for the attribute **hasbeeninteractive**.
- **onrunsuspended**. Function(RunInfo). This function is called when the interactive document composition is suspended by the user. During the callback, RunInfo object is passed. The element identified by the elementid parameter on the Start call contains the last form that was presented to the end user during the interactive document composition.

  If the call does not contain any form, see RunInfo for the attribute **hasbeeninteractive**.
- **onrunfailed**. Function(RunInfo, Error). This function is called when the interactive document composition is completed in an error state. During the callback, RunInfo and Error objects are passed.
- **onerror**. Function(RunInfo, Error). This function is called when an error occurs during the interactive document composition, but the process continues. During the callback, RunInfo and Error objects are passed.

CcmComposerUIAPIV1.Model.RunOptions

The CcmComposerUIAPIV1.Model.RunOptions call returns a RunOptions object that can be passed to the Start call. The call takes a single object as an argument. From this object, the following attributes are registered as options:

- **locale**. Object. The value of the locale.ui attribute of this object is used to determine the user interface language to be used during the interactive document composition. The following values are currently supported: "en" and "nl" for English and Dutch, respectively.
- **tbescripturl**. String. The relative location of the TinyMCE jQuery file that is used to implement the Text Block support.

  The default value for this setting: tinymce/tiny_mce_src.js

- **tbeformatfunctions**. Array. An array of format function objects to be presented to the end user in the formatting dialog in the Text Block Editor. A format function consists of the name attribute and the parameters attribute.
  Example `[{name: 'Concat', parameters: ['prefix']}, {name: 'AsNumber', parameters: ['nr_of_decimals']}]`.

RunInfo

A RunInfo object is passed on the callback function to pass information about the run. A RunInfo object contain values for the following attributes:

- **starturl**. String. The starturl passed to the Start call.
- **sessionid**. String. The sessionid passed to the Start call.
- **jobid**. String. The jobid passed to the Start call.
- **elementid**. String. The elementid passed to the Start call.
- **options**. RunOptions. The options object passed to the Start call.

In case of the onruncompleted callback, the following attribute is also available:

- **hasbeeninteractive**. Boolean. A boolean value that indicates whether any interaction was required to compose the document. Based on this value, the business application may determine whether to provide a button that leads the end user back to the last form of the interactive document composition.

Document

A Document object is passed on the onruncompleted callback. The object offers two methods: one is to retrieve the content of the document, and the other to update it.

The GetContent method accepts one callback parameter:

- **callback**. Function(format, content). This method is called once the content of the document is retrieved from the server. The format parameter contains the format of the result document (.doc, .docx, or .pdf). The content attribute contains the actual content (base64 encoded).

The SetContent method accepts the content as its only parameter:

- **content**. Base64 string. The content that is uploaded to the server to replace the currently stored content of the document.

Error

When an error occurs, an Error object is passed to the onrunfailed or onerror callback. This object contains values for the following attributes:

- **type**. String. "http" or "ui."
- **message**. String. A message used to display the error to the end user.
- **details**. Object. An object containing more details about the error. The exact contents of the object differs between error types.

For type "http," the error.details object contains the following attributes:

- **url**. String. The URL that is accessed.
- **statuscode**. Number. The http status code that is returned.
- **statustext**. String. The status text (if any) that is returned in the http header.
- **responsetext**. String. The response text (if any) that is returned in the response body.

For type "ui," the error.details object contains the following attributes:

- **context**. String. A textual representation that indicates the context in which the error occurred.
- **exception**. Object. The JavaScript exception that is thrown to indicate the error.

Chapter 3

Content management

There are a few limitations to be aware of while designing content for KCM ComposerUI for HTML5.

Unsupported features

The following features of interactive forms are not supported in KCM ComposerUI for HTML5:

- Editable Rich Text Blocks
- Grouping of `FORM` content in a `TABLE` (`BEGINTABLE…ENDTABLE` instruction)
- Key selection

Content Wizards

While KCM ComposerUI for ASP.NET and KCM ComposerUI for J2EE present a Content Wizard as a sequence of Forms, KCM ComposerUI for HTML5 presents a Content Wizard as a single and integrated user interface.

In this regard, the following additional requirement applies on the scoping of QForms on the Content Wizard and the corresponding Data Backbone nodes:

- Any Fields that are referred to from these QForms have to be available at the Data Backbone level that corresponds to the node on which the QForm is operating.
- If the Fields are not available, this error message appears.

```
 DYN9000: Unable to resolve fieldset ... in
dynamic object ... KCM CompuserUI requires that QForms are
explicitly attached to a node where all variables are
available.
```

In this case, a possible solution would be to introduce an additional section under the repeating Data Backbone node in the Content Wizard and then configure the QForm on this section.

Chapter 4

Troubleshooting

This chapter gives troubleshooting tips that may be useful if you encounter an issue while using the KCM ComposerUI for HTML5.

Text Block Editor error handling

When editing a Text Block in ComposerUI HTML5, a TinyMCE-based editor is loaded. For various reasons loading may fail. In this case, the following error message is prompted after a timeout.

```
A timeout occurred while loading the editor. The application
   failed to load the text block editor, or loading is slow. Please refer to the
   JavaScript API Manual for ComposerUI for guidance on troubleshooting this
   issue.
```

ComposerUI for HTML5 is integrated in a page of the customer's business application (for more information, see Introduction). This page should refer to a TinyMCE distribution (for more information, see Dependencies). This distribution contains KCM specific plug-ins. The latest version of this distribution is available on the server at:

https://<kcm server>/proxy/tinymce (or **http://<kcm server>:8080/proxy/tinymce**, if SSL is not enabled)

The version on the server always matches the KCM version. Furthermore, language packs are applied to this version.

If TinyMCE is consumed from another location, the following may cause problems when loading the Text Block editor:

- The TinyMCE distribution contains old versions of the plugins that do not match the KCM version on the server.
- ComposerUI is running with a custom language based on a language pack. This language pack is not applied on the TinyMCE distribution.

Either of these problems causes loading of the editor to fail and thus the timeout to occur.

In both cases, we strongly recommend that you refer to the TinyMCE distribution on the server. The example.html page that resides at the following locations contains more information about this subject:

https://<kcm server>/start/example/example.html (or **http://<kcm server>:8080/start/example/example.html**, if SSL is not enabled)

PDF preview does not show scroll bars

During the composition of a Document Pack Template, if the PDF preview in the review pane of ComposerUI for HTML5 does not show scroll bars, and the PDF cannot be displayed in full size, ensure that the respective top-level container element has a fixed height. The top-level container element is the element identified by elementid passed to the CcmComposerUIAPIV2.Run.Start call. You can customize the element by changing the .kccmroot class.