Kofax Customer Communications Manager

5.0

JavaScript API Manual for ComposerUI

KOFAX
from Lexmark

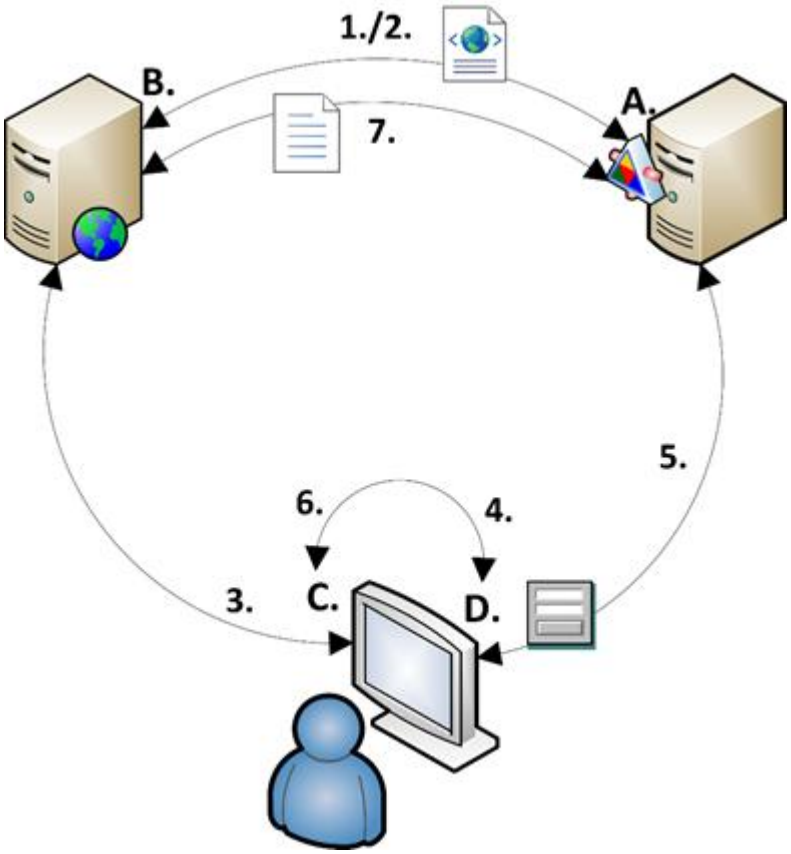
Contents

Introduction

This document describes how CCM ComposerUI for HTML5 can be integrated in a web application, using the CCM ComposerUI JavaScript API. This document targets (web) developers of any business application that requires the integration of interactive document composition.

The section Content Management contains a few important notes on the relation between content elements and the CCM ComposerUI. This section targets content managers that use the CCM Designer for Web to design interactive documents.

Architecture



The following components are involved in interactive document composition:

A. CCM Core
B. The business application (server side)
C. The business application (web browser, client side)
D. The CCM ComposerUI JavaScript API, loaded in a page of the business application (web browser, client side)

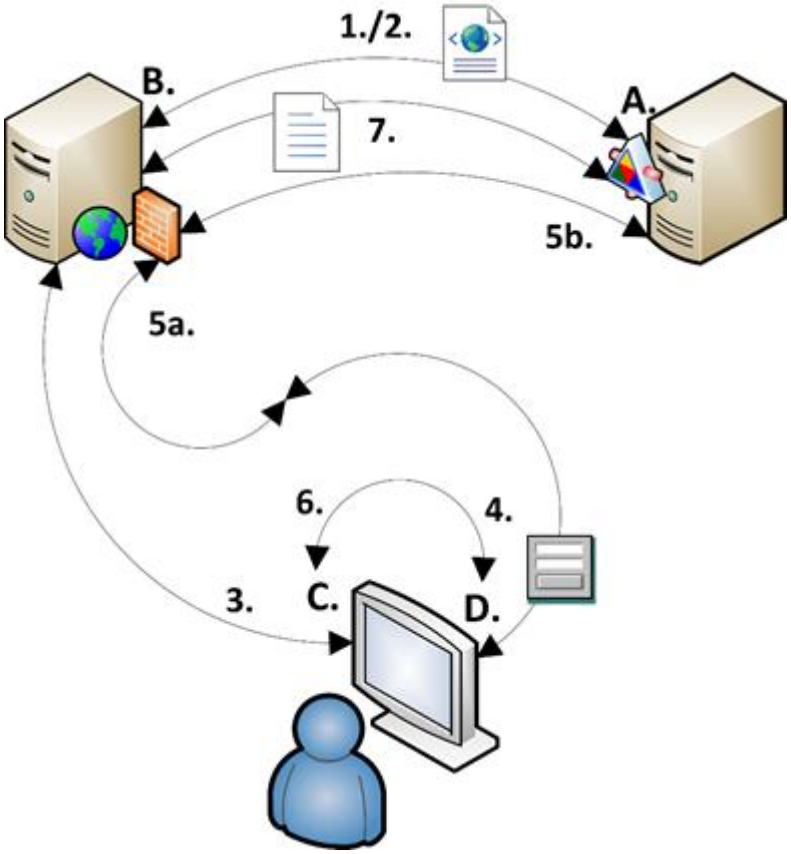
The process of interactive document composition consists of the following steps:

1. The business application (server side) calls CCM Core through its web services interface. The specific call will either be ComposeDocxInteractiveStart or ComposePdfInteractiveStart, depending on the required output. On this call it will pass a.o.:
   a. A reference to the object (template or letterbook) in the context of the CCM Designer for Web
   b. The data for the interactive document composition run
2. CCM Core will register a session for the run and return:
   a. A url
   b. A session identifier
3. The business application will pass this information to its client side
4. The business application will call the CCM ComposerUI JavaScript API, passing a.o:
   a. The url
   b. The session identifier
   c. An identification of an HTML-element on the page
   d. A number of JavaScript callback functions

5. The CCM ComposerUI JavaScript API will connect to CCM Core. Based on the retrieved information it will present all interaction required for composition of the document to the end user. All interaction will be presented inside the HTML-element that was indicated by the call to the API.
6. Whenever the interactive process finishes (either successful or in error state) an appropriate callback function is called by the API. This allows the business application to pick up the process and e.g. (in case of success) notify its server side that the document has been composed.
7. The business application (server side) calls CCM Core through its web services interface. The specific call will either be ComposeDocxInteractiveGet or ComposePdfInteractiveGet, matching the previous call. On this call it will pass the session identifier. The call will return the composed document.

A word on CORS

In the picture above, CCM Core is accessed directly from the browser, whereas the page from which it is accessed resides on the web server of the business application. This is called Cross-origin resource sharing (CORS) and is only allowed under certain conditions. In Microsoft Internet Explorer 10 and up, e.g., 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 (always) met in the environment in which the business application is deployed, it is strongly recommended to route the HTTP-traffic to CCM Core through a proxy that shares its base url with the business application. In this case, CORS-related restrictions do not

apply.

API

The current version of the API is CcmComposerUIAPIV2. For CcmComposerUIAPIV1, see the paragraph Previous versions (page 13) below.

The description of the API will assume the url and session identifier, described in the previous paragraph, to be available.

Dependencies

The JavaScript API can be downloaded from the following location on a CCM Core installation: http://<ccm server>:8081/start/home.html, where <ccm server> is the hostname of the server. The JavaScript is shipped with a corresponding CSS file, which can be modified if required.

The API depends on the following components:

- jQuery (version 1.8 or higher)
- jQueryUI (version 1.10 or higher)
- moment.js (version 2.6 or higher)
- TinyMCE, including a number of plugins for specific CCM Text Block support. A full tree can be downloaded from the start page as well.

Furthermore, 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, **CcmComposerUIAPIV2.Run.Start**, with the following parameters:

- **starturl.** *String.* An absolute url to CCM Core, which is the absolute version of the relative url that was retrieved through the web services call. This url will be accessed from the browser on the client machine, so the base of this url will have to make sense in the specific network environment (taking into account proxies, etc.).
- **sessionid.** *String.* The session identifier that was retrieved through the web services call.
- **jobid.** *String.* An identifier of the specific run. This will only be used to identify the run in logs and on callbacks.
- **elementid.** *String.* The HTML id of the element on the page that will contain the user interaction for the document composition process.
- **callbacks.** *CcmComposerUIAPIV2.Model.RunCallbacks.* An object that exposes a number of callbacks that will be called to notify the business application of an event. See the next paragraph for more information.
- **options.** *CcmComposerUIAPIV2.Model.RunOptions.* An object containing some additional options. See the next paragraph for more information.

The call will return true if the run was started successfully, false otherwise. Any subsequent information will be passed through the callbacks.

Along with this, the API exposes **CcmComposerUIAPIV2.Version.Get**. This returns a Version

object with the following attributes:

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

This information will be useful, e.g. in the context of a support call.

Objects

The following objects can be created through the API:

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

The following objects are passed on 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 argument. From this object the following attributes are registered as callback:

- **onruncompleted.** *function(RunInfo, Result).* This function will be called when the interactive document composition process has been completed successfully. On the callback a RunInfo and a Result object are passed (see below). The element identified by the elementid parameter on the Start call will contain the last form (if any – see *hasbeeninteractive* below) that was presented to the end user during interactive document composition.
- **onrunsuspended.** *function(RunInfo).* This function will be called when the interactive document composition process has been suspended by the user. On the callback a RunInfo object is passed (see below). The element identified by the elementid parameter on the Start call will contain the last form (if any – see *hasbeeninteractive* below) that was presented to the end user during interactive document composition.
- **onrunfailed.** *function(RunInfo, Error).* This function will be called when the interactive document composition process completes in error state. On the callback a RunInfo and an Error object are passed (see below).
- **onerror.** *function(RunInfo, Error).* This function will be called when an error occurs during the interactive document composition process (but the process continues). On the callback a RunInfo and an Error object are passed (see below).

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 argument. From this object the following attributes are registered as options:

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

text block support. The default value for this setting is *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 a 'name' attribute (*String*) and a 'parameters' attribute (*array of String*), so e.g. `[{name: 'Concat', parameters: ['prefix']}, {name: 'AsNumber', parameters: ['nr_of_decimals']}]`.
- **stylemap.** *object.* A map of key/value-pairs that will determine the actual class names that are used for a number of logical class names (described below). This allows the caller to determine the look and feel of a well-known subset of the UI using an application-specific css file.

Stylemap

The following keys can be passed on the stylemap parameter. If a value is available for a key, this value will be applied as a class name to the elements that are described below. Otherwise, a default value is used.

key	default value	description
active	kccmactive	Used to mark elements that are active (e.g. the text blocks in the selection that is currently presented in the right pane).
highlight	kccmhighlight	Used to highlight elements (e.g. the text block that the user hovers in the right pane).
error	kccmerror	Used to mark elements that indicate error state.
selected	kccmselected	Used to mark selected elements (e.g. the selected item when reviewing a document pack).
button	kccmbutton	General button, used for form submission (+for editing an 'editabletextblock' question).
submitbutton	kccmsubmitbutton	General button, used for form submission
submitbuttonok	kccmsubmitbuttonok	"Ok" button, advancing interaction to the next form
submitbuttonback	kccmsubmitbuttonback	"Back" button, returning interaction to the previous form
submitbuttonupdate	kccmsubmitbuttonupdate	"Update" button, applying the value of a field to the current document
cwbutton	kccmcwbutton	Button to open selection pane (textblock 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.

key	default value	description
tooltip	kccmtooltip	Tooltip that is presented if the user hovers a help icon for a question. Remark: 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 title, if available.
group	kccmgroup	Group div containing any number of grouped questions/groups.
grouphead	kccmgrouphead	Header div containing the name of the group.
cwdocument	kccmcwdocument	The canvas in which the content of the document is presented.
cwsidebar	kccmcwsidebar	The sidebar that is shown alongside the canvas.
cwfield	kccmcwfield	A field inside the text blocks that fill the canvas.
cwfieldeditable	kccmcwfieldeditable	Annotation of cwfield for fields that can be modified by clicking it.
textblockpreview	kccmtextblockpreview	A preview of a text block, which is shown for ‘editabletextblock’, ‘textblocksingleselect’ and ‘textblockmultiselect’ questions.
textblockpreviewfield	kccmtextblockpreviewfield	A field inside a text block preview.
textblockeditor	kccmtextblockeditor	The text block editor, used to present editing facilities for text blocks that have been marked as editable.
questionlabel	kccmquestionlabel	Question label.
questionlabelerror	kccmquestionlabelerror	The label of a question that has an invalid answer.
letterbookfolder	kccmletterbookfolder	List of elements within a folder in the letterbook tree.
letterbookfoldertitle	kccmletterbookfoldertitle	Folder list element (node) in the letterbook tree.
letterbooktemplate	kccmletterbooktemplate	Template list element (leave) in the letterbook tree.

key	default value	description
letterbooktemplatelink	kccmletterbooktemplatelink	Template link that allows the user to select a template.
statictext	kccmstatictext	Static text on a form (e.g. used for some more extensive explanation about a number of questions).
textquestion	kccmtextquestion	Question control that is presented to the user to answer a specific type of question.
textareaquestion	kccmtextareaquestion	Question control that is presented to the user to answer a specific type of question.
numberquestion	kccmnumberquestion	Question control that is presented to the user to answer a specific type of question.
boolquestion	kccmboolquestion	Question control that is presented to the user to answer a specific type of question.
datequestion	kccmdatequestion	Question control that is presented to the user to answer a specific type of question.
timequestion	kccmtimequestion	Question control that is presented to the user to answer a specific type of question.
filequestion	kccmfilequestion	Question control that is presented to the user to answer a specific type of question.
singleselectquestion	kccmsingleselectquestion	Question control that is presented to the user to answer a specific type of question.
radiosingleselectquestion	kccmradiosingleselectquestion	Question control that is presented to the user to answer a specific type of question.
multiselectquestion	kccmmultiselectquestion	Question control that is presented to the user to answer a specific type of question.
textblocksingleselectquestion	kccmtextblocksingleselectquestion	Question control that is presented to the user to answer a specific type of question.
textblockmultiselectquestion	kccmtextblockmultiselectquestion	Question control that is presented to the user to answer a specific type of question.
editabletextblockquestion	kccmeditabletextblockquestion	Question control that is presented to the user to answer a specific type of question.
editablerichtextblockquestion	kccmeditablerichtextblockquestion	Question control that is

key	default value	description
		presented to the user to answer a specific type of question.
reviewheader	kccmreviewheader	The header that is shown when reviewing a document pack.

RunInfo

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

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

In case of the *onruncompleted* callback the following attribute will also be available:

- **hasbeeninteractive.** *Boolean.* A boolean value that indicates whether any interaction was required to compose the document. Based on this, e.g., 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.** Either "document" or "documentpack", depending on whether the document composition resulted in a single document or a pack of documents.
- **object.** The actual result. Currently only provided if type="document", in which case the attribute exposes an object of type Document (see below); *null* otherwise.

Document

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

The GetContent method accepts one callback parameter:

- **callback.** *function(format, content).* This method is called as soon as the content of the document is retrieved from the server. The *format* parameter will contain the format of the result document ('doc', 'docx' or 'pdf'). The *content* will contain the actual content (base64 encoded).

The SetContent method accepts the content as its only parameter:

- **content.** *base64String.* The content that will be uploaded to the server to replace the currently stored content of the document.

Error

Whenever an error occurs, an Error object is passed to either the *onrunfailed* or the *onerror* callback. This object is guaranteed to contain sensible values for the following attributes:

- **type.** *String.* Currently either 'http' or 'ui'.
- **message.** *String.* A message that may be 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 will differ between error types.

For type 'http' the *error.details* object will contain:

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

For type 'ui' the *error.details* object will contain:

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

Editing the result

In many scenarios, editing of a Word document – immediately after it has been composed – is a requirement of the business process. This is not part of the interactive document composition process, which starts with an invocation of the *Start* call and finishes with a call to the *onruncompleted* callback. The API does, however, contain facilities that allow for a seamless integration of editing in the process.

As described in its section, the Document object that is passed on the *onruncompleted* callback exposes *GetContent* and *SetContent* methods. Through these the content of the result document can be retrieved and updated. A subsequent call to the ComposeDocxInteractiveGet will then return the updated document.

One way of offering the content for editing is to use an ActiveX control. ActiveX is a technique that has many drawbacks, but can still provide useful functionality in many contexts. An ActiveX control that supports editing of base64 Word content is available on a Kofax CCM installation and can be downloaded from the start page. The example below shows how the ActiveX-control can be used to offer interactive document composition with post-composition editing of the resulting document.

Previous versions

CcmComposerUIAPIV1

Calls

The CcmComposerUIAPIV1 API exposes one main call, **CcmComposerUIAPIV1.Run.Start**, with the following parameters:

- **starturl.** *String.* An absolute url to CCM Core, which is the absolute version of the relative url that was retrieved through the web services call. This url will be accessed from the browser on the client machine, so the base of this url will have to make sense in the specific network

environment (taking into account proxies, etc.).

- **sessionid.** *String.* The session identifier that was retrieved through the web services call.
- **jobid.** *String.* An identifier of the specific run. This will only be used to identify the run in logs and on callbacks.
- **elementid.** *String.* The HTML id of the element on the page that will contain the user interaction for the document composition process.
- **callbacks.** *CcmComposerUIAPIV1.Model.RunCallbacks.* An object that exposes a number of callbacks that will be called to notify the business application of an event. See the next paragraph for more information.
- **options.** *CcmComposerUIAPIV1.Model.RunOptions.* An object containing some additional options. See the next paragraph for more information.

The call will return true if the run was started successfully, false otherwise. Any subsequent information will be passed through the callbacks.

Along with this, the API exposes **CcmComposerUIAPIV1.Version.Get**. This returns a Version object with the following attributes:

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

This information will be useful, e.g. in the context of a support call.

Objects

The following objects can be created through the API:

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

The following objects are passed on 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 argument. From this object the following attributes are registered as callback:

- **onruncompleted.** *function(RunInfo, Document).* This function will be called when the interactive document composition process has been completed successfully. On the callback a RunInfo and a Document object are passed (see below). The element identified by the elementid parameter on the Start call will contain the last form (if any – see *hasbeeninteractive* below) that was presented to the end user during interactive document composition.
- **onrunfailed.** *function(RunInfo, Error).* This function will be called when the interactive document composition process completes in error state. On the callback a RunInfo and an Error object are passed (see below).
- **onerror.** *function(RunInfo, Error).* This function will be called when an error occurs during the interactive document composition process (but the process continues). On the callback a RunInfo and an Error object are passed (see below).

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 argument. From this object the following attributes are registered as options:

- **locale.** *object.* The value of the locale.ui attribute of this object will be used to determine the user interface language to be used during interactive document composition. Currently the following values are supported: 'en' and 'nl' (for English and Dutch, respectively).
- **tbescripturl.** *String.* The relative location of the TinyMCE jQuery file that is used to implement text block support. The default value for this setting is *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 a 'name' attribute (*String*) and a 'parameters' attribute (*array of String*), so e.g. `[{name: 'Concat', parameters: ['prefix']}, {name: 'AsNumber', parameters: ['nr_of_decimals']}]`.

RunInfo

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

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

In case of the *onruncompleted* callback the following attribute will also be available:

- **hasbeeninteractive.** *Boolean.* A boolean value that indicates whether any interaction was required to compose the document. Based on this, e.g., 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.

Document

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

The GetContent method accepts one callback parameter:

- **callback.** *function(format, content).* This method is called as soon as the content of the document is retrieved from the server. The *format* parameter will contain the format of the result document ('doc', 'docx' or 'pdf'). The *content* will contain the actual content (base64 encoded).

The SetContent method accepts the content as its only parameter:

- **content.** *base64String.* The content that will be uploaded to the server to replace the currently stored content of the document.

Error

Whenever an error occurs, an Error object is passed to either the *onrunfailed* or the *onerror* callback. This object is guaranteed to contain sensible values for the following attributes:

- **type.** *String.* Currently either 'http' or 'ui'.
- **message.** *String.* A message that may be 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 will differ between error types.

For type 'http' the *error.details* object will contain:

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

For type 'ui' the *error.details* object will contain:

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

Example

The start page (http://<ccm server>:8081/start/home.html) provides access to an example integration of the JavaScript API (Links-> CCM ComposerUI for HTML5 ->Test). This example runs a fixed template from an example CCM Designer for Web project. It can be used to test the installation. It can also be downloaded from the start page and used as a reference for an integration in a business application. It provides information on how it achieves the integration in inline comments in its example.html file.

Content Management

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

Unsupported features

The following features of interactive forms are not (or not yet) supported by CCM ComposerUI for HTML5:

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

Content Wizards

Where CCM ComposerUI for ASP.NET and CCM ComposerUI for J2EE present a Content Wizard as a sequence of forms, CCM ComposerUI for HTML5 presents a Content Wizard as a single, integrated user interface. This puts an extra requirement on the scoping of QForms on the nodes in the Content Wizard (and the corresponding nodes in the Data backbone): any fields that are referred from these QForms have to be available at the Data backbone level that corresponds to the node on which the QForm is operating. If this is not the case, this will result in the following error message:

```
DYN9000: Unable to resolve fieldset ... in dynamic object ... CCM 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 configure the QForm on this section.