Kofax Communications Manager

ComposerUI for ASP.NET and J2EE Customization Guide

Version: 5.3.0

Date: 2019-05-28

KOFAX

Table of Contents

Preface

This guide describes the customization options for KCM ComposerUI for ASP.NET and J2EE, introduces the structure of the XSL transformation that produces the KCM ComposerUI web forms and discovers an alternative for the HTML output.

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

You can integrate KCM ComposerUI for ASP.NET and KCM ComposerUI for J2EE in many contexts that support customization. This guide describes the customization options in detail and serves as a supplement to the *KCM ComposerUI for ASP.NET and J2EE Developer's Guide*. It also gives you information about div-oriented output and the underlying JavaScript.

This guide assumes basic knowledge of the concepts of KCM ComposerUI and of XSLT.

Chapter 2

Web forms: XSLT

During interactive document composition the end user is presented with a number of forms. These forms are defined in XForms format by the KCM Master Template that runs on KCM Core. Such a form definition is transformed to a web form by an XSL transformation on KCM ComposerUI.

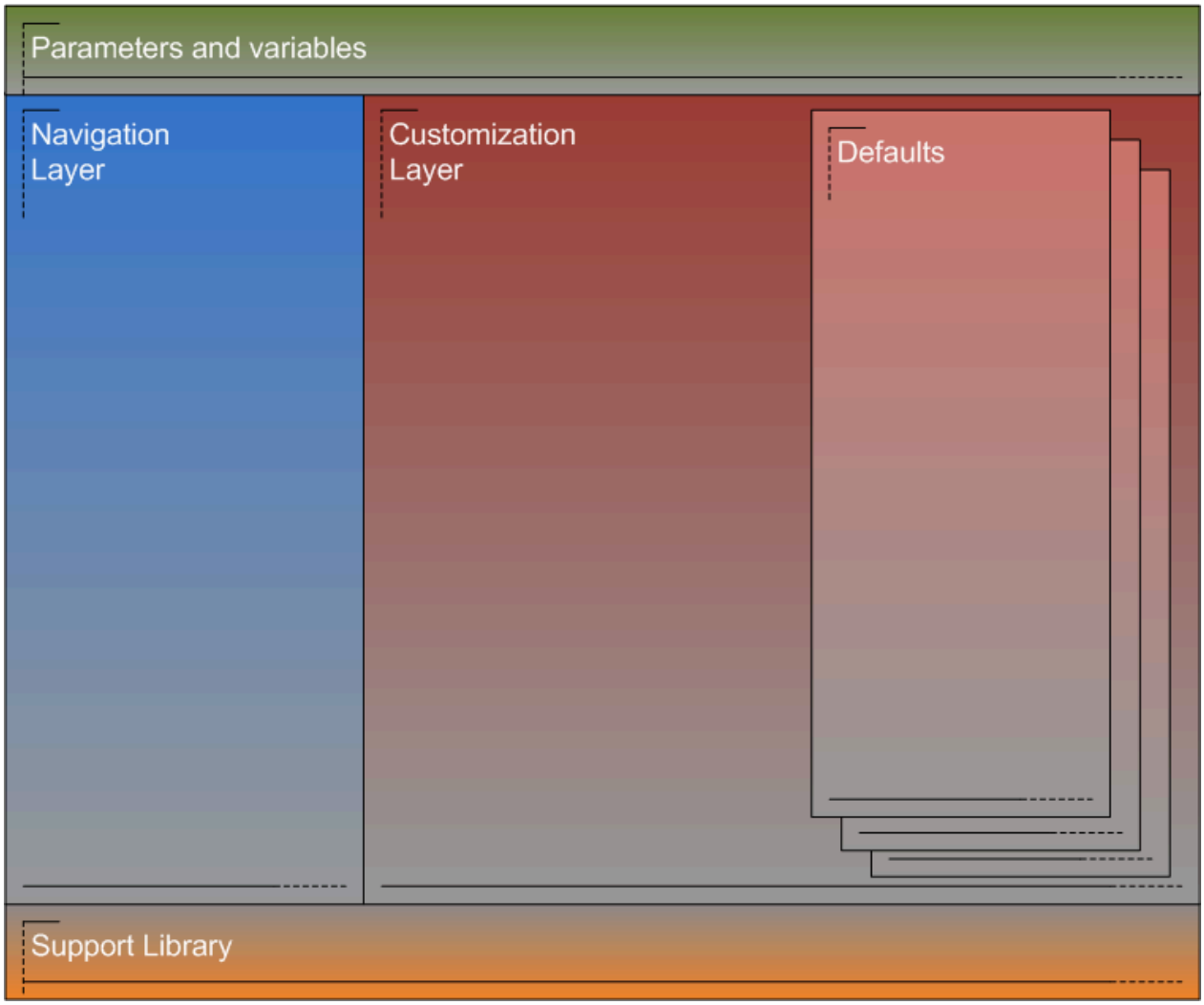
XSLT

The XSL transformation that transforms the XForms form definitions to web forms can be overridden. Up to version 3.5.12 of KCM ComposerUI, XSL transformations were not modular. The XSL transformation could have been either overridden as a whole, or not at all, which had an important impact on updates. If an update of KCM ComposerUI involved new functionality, part of which was implemented in the XSL transformation, this functionality would have been shielded because of customizations to the XSL transformation. It demonstrated an increasing need for merges with each update.

From version 3.5.12 overrides may be applied to well-defined parts of the XSL transformation. This chapter describes the modular structure of the XSL transformation.

Note This structure currently applies to the transformation file interact.xsl, namely the transformation that produces the KCM ComposerUI web forms.

General structure

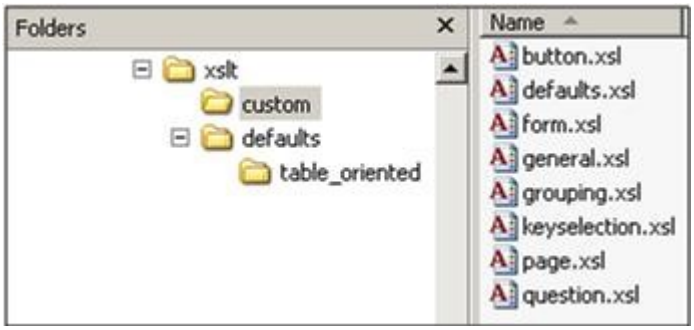


The general structure of the XSL transformation consists of:

- **Parameters and variables**: A set of well-defined global parameters and variables to use as part of the customization.
- **Support library**: A set of well-defined named templates for use as part of the customization. These templates cannot be overridden.
- **A navigation layer**: A set of well-defined templates that operate on the XForms. These templates shield the customizable parts of the XSLT from the details of the XForms format. The templates in the navigation layer do not produce any output and cannot be overridden.
- **A customization layer**: A set of well-defined templates called from the navigation layer to produce an isolated chunk of output. For this purpose, they can use any functionality defined in the other layers. These templates can be overridden.

- **Defaults**: A default implementation for each template from the Customization layer. These defaults are called by the templates in the Customization layer but can also be called from overrides of these templates. Currently, only one set of defaults exists, producing the classic table-oriented KCM ComposerUI output. These defaults cannot be overridden.

Files



The structure above is implemented in a number of files, which reside in the XSLT subfolder of the KCM ComposerUI installation. This folder contains the following subfolders:

- **Custom**: This is the only folder with files that may be modified. Declaration and implementation of a customizable template in one of these files implies that the template is overridden. The file defaults.xsl is responsible for selecting an implementation of the defaults from the subfolder Defaults.
- **Defaults**: This folder contains the defaults as described above. These are implemented in the file defaults.xsl, which resides in another subfolder. Currently, one subfolder is named table_oriented, which produces the old table-oriented output, and one subfolder named div_oriented, which produces the new output described in the next chapter.

At the top level, four files implement the layers mentioned earlier:

- **varsandparams.xsl** for parameters and variables
- **support.xsl** for the support library
- **navigation.xsl** for the navigation layer
- **overridable.xsl** for the customization layer

The last file is interact.xsl, which is the main .xsl file. This file imports the other templates. KCM ComposerUI applies it to the XForms.

The interact.xsl calls templates from navigation.xsl, and in turn it calls templates implemented in overridable.xsl. These templates can be overridden by simply implementing them in one of the custom \ *.xsl files. The implementation of those templates in one of the custom\ *.xsl files may call the _default templates from the defaults\ *.xsl files. By default the code, which is commented in the custom\ *.xsl files, calls the corresponding templates in default\ *.xsl files.

Appendix B contains an example of the customization flow presented in the Example below.

Note Do not implement overrides by modifying overridable.xsl. Instead, you must define overrides in the files that reside in the custom subfolder.

Templates

Appendix A lists all templates that define the guaranteed interfaces between the different layers of the XSL transformation. There are three template categories:

- Templates defined in **Navigation Layer**. These are templates that can be applied to a given context in the XForms.
- Templates defined in **Customization Layer**. These are named templates that are used to produce an isolated bit of output. For each template with name X in the custom layer, there is a template named X_default in the defaults.
- Templates defined in **Support Library**. These named templates implement some useful support functionality.

Most of the named templates require a given XForms context. In other words, they can only be called if the current context node is the right type.

Appendix B presents an example of the flow of control between the different layers. Navigation starts by applying the main template itp:interact in the navigation layer. This calls the template producePage in the Customization layer, which produces the page output. This template contains a simple call to its equivalent in the defaults, which may call templates in either the Navigation or the Customization layer and uses templates from the Support Library in the process. Templates defined in \ *custom*\ **.xsl* override templates in overridable.xsl.

Passing information

Information is passed on the calls between the different templates. This information is bundled in so-called node sets. The signature of the different node sets is described in Appendix C.

Along with the node sets, a "custom" parameter is passed on between the templates. The navigation layer only passes the custom parameter, but does not interpret it. The parameter can be used to distinguish between different traversals of the XForms structure, if that is required as part of the customization. The Example in the next section illustrates the intended use of the "custom" parameter.

Example

Part of the installation of KCM ComposerUI is a custom1.zip file, which contains an example extension to the KCM ComposerUI application. This extension serves to illustrate XSLT customization. The customization in the custom1 application adds buttons at the top of the forms, so that the end user does not have to scroll down to submit a long form.

The custom1 application is not installed by default. To install it, we recommend these steps:

- Go to the configuration page of your KCM ComposerUI installation. See the Configuration chapter of the *KCM ComposerUI Developer's Guide.*
- In the field Application Name, fill out **"custom1"** and click **Submit**.
- In the folder OnLine application, a subfolder custom1 is created. Copy the contents of the subfolder sample2 to this location.
- Unzip the contents of the custom1.zip file to the subfolder. Make sure the unzip does not introduce an extra level custom1; a subfolder XSLT should occur immediately below the existing folder custom1.
- For KCM ComposerUI ASP.NET only, go back to the configuration page and click the link **Deploy** next to the application custom1.

The application custom1 is an addition to the application sample2 that is installed by default. It adds the following files:

- *\css\topbuttons.css.* A CSS file with a small number of extra style definitions for the new buttons.
- *\xslt\custom\button.xsl*. An XSL file overrides the templates produceButtons, produceButton, and produceButtonLabel from the Customization Layer. These templates produce the actual buttons.
- *\xslt\custom\form.xsl*. An XSL file overrides the template produceFormBody from the Customization Layer. This template positions the buttons at the top of the form.
- *\xslt\custom\page.xsl*. An XSL file overrides the template producePageCSS from the Customization Layer. This template includes the additional topbuttons.css stylesheet.

The other files in the folder \ xslt\ custom are irrelevant. They merely serve as a starting point for other customizations. Uncomment a template to override it.

Important general features illustrated by the customization include:

- The use of the custom parameter to distinguish between traversals.
- Reuse of default behavior by calling the _default templates to avoid unnecessary duplication of code.

The override of the template produceFormBody looks similar to the following sample:

```
<xsl:template name="produceFormBody">
  <xsl:param name="interactinfo"/>
  <xsl:param name="custom"/>

  <!-- an additional table with a single row of buttons at the top of the form -->
  <tr>
    <td>
      <xsl:call-template name="produceButtons">
        <xsl:with-param name="interactinfo" select="$interactinfo"/>
        <xsl:with-param name="custom" select="'topbuttons'"/>
```

```
      </xsl:call-template>
    </td>
  </tr>
  <xsl:call-template name="produceFormBody_default">
    <xsl:with-param name="interactinfo" select="$interactinfo"/>
    <xsl:with-param name="custom" select="$custom"/>
  </xsl:call-template>
</xsl:template>
```

This code introduces an extra row with buttons above the standard content of the form. The content of this row is determined by a call to the template produceButtons, bringing in an extra traversal of the buttons in the XForms. To distinguish this traversal from the standard traversal that produces the standard buttons, the value topbuttons is passed for the "custom" parameter. This preceding code illustrates the intended use of this parameter.

The override of the template produceButtons looks similar to the following sample:

```
<xsl:template name="produceButtons">
    <xsl:param name="interactinfo"/>
    <xsl:param name="custom"/>

    <xsl:choose>
      <xsl:when test="$custom='topbuttons'">
        <div class="topbuttons">
          <!-- only produce the ok and the back button (css will order) -->
          <xsl:call-template name="applyNamedButtons">
            <xsl:with-param name="names" select="'ok back1'"/>
            <xsl:with-param name="custom" select="$custom"/>
          </xsl:call-template>
        </div>
      </xsl:when>
      <xsl:otherwise>
        <xsl:call-template name="produceButtons_default">
          <xsl:with-param name="interactinfo" select="$interactinfo"/>
          <xsl:with-param name="custom" select="$custom"/>
        </xsl:call-template>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

Based on the value of the custom parameter, the template either produces the new buttons or falls back on default behavior. At the top of the form only the buttons OK and possibly the Back are produced inside a div with class "topbuttons". This is done by:

- Calling the template applyNamedButtons from the Support Library
- Passing identifications of the buttons to be produced (space separated)
- Passing the value topbuttons for the custom parameter

The template applyNamedButtons passes this value on to the template produceButton. The order of the passed identifiers determines the order of the produced corresponding buttons. Buttons are only produced when they are part of the XForms.

Even though the Back button is intended to be presented to the left of the OK button, the latter is presented first, so OK will become the default button on the form. The positioning of the buttons on the page is arranged in the cascading stylesheet.

The override of the template produceButton looks similar to the following sample:

```
  <xsl:template name="produceButton">
    <xsl:param name="buttoninfo"/>
```

```
   <xsl:param name="custom"/>
   <xsl:choose>
     <xsl:when test="$custom='topbuttons'">
       <button id="{$buttoninfo/submission}" name="{$buttoninfo/submission}"
type="submit" class="{$buttonclassbase}">
         <xsl:attribute name="onClick">
           setSubmission('<xsl:value-of select="$buttoninfo/submission"/>');
         </xsl:attribute>
         <xsl:call-template name="produceButtonLabel">
           <xsl:with-param name="buttoninfo" select="$buttoninfo"/>
           <xsl:with-param name="custom" select="$custom"/>
         </xsl:call-template>
       </button>
     </xsl:when>
     <xsl:otherwise>
       <xsl:call-template name="produceButton_default">
         <xsl:with-param name="buttoninfo" select="$buttoninfo"/>
         <xsl:with-param name="custom" select="$custom"/>
       </xsl:call-template>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:template>
```

Again, the custom parameter determines whether or not to fall back on default behavior. The new buttons are produced directly below the div that is produced by the template produceButtons. The button label is produced by calling the template produceButtonLabel, again passing the value topbuttons for the custom parameter.

The override of the produceButtonLabel template looks similar to the following sample:

```
 <xsl:template name="produceButtonLabel">
   <xsl:param name="buttoninfo"/>
   <xsl:param name="custom"/>

   <xsl:choose>
     <xsl:when test="$custom='topbuttons'">
       <xsl:choose>
         <xsl:when test="$buttoninfo/submission='ok'">
           <xsl:text disable-output-escaping="yes">&gt;&gt;</xsl:text>
         </xsl:when>
         <xsl:when test="$buttoninfo/submission='back1'">
           <xsl:text disable-output-escaping="yes">&lt;&lt;</xsl:text>
         </xsl:when>
         <xsl:otherwise>
           <xsl:call-template name="produceButtonLabel_default">
             <xsl:with-param name="buttoninfo" select="$buttoninfo"/>
             <xsl:with-param name="custom" select="$custom"/>
           </xsl:call-template>
         </xsl:otherwise>
       </xsl:choose>
     </xsl:when>
     <xsl:otherwise>
       <xsl:call-template name="produceButtonLabel_default">
         <xsl:with-param name="buttoninfo" select="$buttoninfo"/>
         <xsl:with-param name="custom" select="$custom"/>
       </xsl:call-template>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:template>
```

Again, the custom parameter determines whether or not to fall back on default behavior. The buttons OK and Back are respectively represented by >> and <<.

The override of the template producePageCSS looks similar to the following sample:

```
<xsl:template name="producePageCSS">
  <xsl:param name="interactinfo"/>
  <xsl:param name="custom"/>
  <xsl:call-template name="producePageCSS_default">
    <xsl:with-param name="interactinfo" select="$interactinfo"/>
    <xsl:with-param name="custom" select="$custom"/>
  </xsl:call-template>
  <link rel="stylesheet" type="text/css" href="css\topbuttons.css"/>
</xsl:template>
```

This template simply adds a reference to the topbuttons.css stylesheet, again using the default behavior to produce the standard set of references. The template topbuttons.css contains a small number of style instructions, to ensure that the buttons are rendered a bit smaller and that the OK button is presented to the right of the Back button.

A visual representation of these customizations appears in Appendix B.

Chapter 3

Web Forms: new output

Version 3.5.15 of KCM ComposerUI Server introduces an alternative for the old, table-oriented HTML output of the KCM ComposerUI forms. This alternative has the following features:

- The output is XHTML-compliant.
- The output is div-oriented, allowing for a more flexible layout through CSS.
- The underlying JavaScript is open and documented, and defines an extensibility model. This means, customization of KCM ComposerUI through JavaScript can be done in a well-defined an maintainable way.
- The underlying JavaScript capitalizes on the power of jQuery (http://www.jquery.com).

This chapter describes the new output, the underlying JavaScript library and the way in which the extensibility model can be used. This description is currently limited to the web forms produced by KCM ComposerUI, namely, the output produced by the file interact.xsl.

Switching between the old and the new output

KCM ComposerUI Server produces the div-oriented output by default. If required, this can be changed by putting a custom file *xslt\ custom\ defaults.xsl* in your KCM ComposerUI application. To activate the legacy table-oriented output, make sure that it contains:

```
<xsl:import href="..\defaults\table_oriented\defaults.xsl"/>
```

The same applies to the output of the KCM ComposerUI letterbook. The file *xslt\ custom \ defaults_letterbook.xsl* includes *xslt\ defaults\ div_oriented\ defaults.xsl*, rather than *xslt\ defaults\ table_oriented\ defaults.xsl*. This can be changed by putting a custom file *xslt\ custom\ defaults_letterbook.xsl* in your KCM ComposerUI application. To activate the old table-oriented output, make sure that it contains:

```
<xsl:import href="..\defaults\table_oriented\defaults_letterbook.xsl"/>
```

The file newsample.zip is included in your KCM ComposerUI installation, but it is not installed by default. To install the file:

1. Go to the configuration page of your KCM ComposerUI installation. See the configuration chapter in the *KCM ComposerUIDeveloper's Guide*.
2. In the field Application Name, fill out "newsample" and click Submit.
3. In the OnLine application folder, a subfolder newsample is created.
4. Unzip the contents of the newsample.zip file to the subfolder. Make sure the unzip does not introduce an extra level newsample; subfolders CSS and XSLT should occur immediately below the existing folder newsample.

5. For KCM ComposerUI ASP.NET only, go back to the configuration page and click the link Deploy next to the application newsample.

HTML

The HTML output is XHTML-compliant and its layout uses divs and spans, rather than tables. The XHTML produced by the XSL transformation is listed in Appendix D. It produces rather basic structures, which are lifted by JavaScript manipulation during loading of the page.

CSS

By default, two cascading style sheets are included in the HTML page:

- A general .css that implements a jQuery theme, which by default is UI-lightness.
- A specific .css that applies directly to the KCM ComposerUI forms layout.

Appendix D presents the XHTML produced by the new XSL transformation and the CSS classes attached to the different tags.

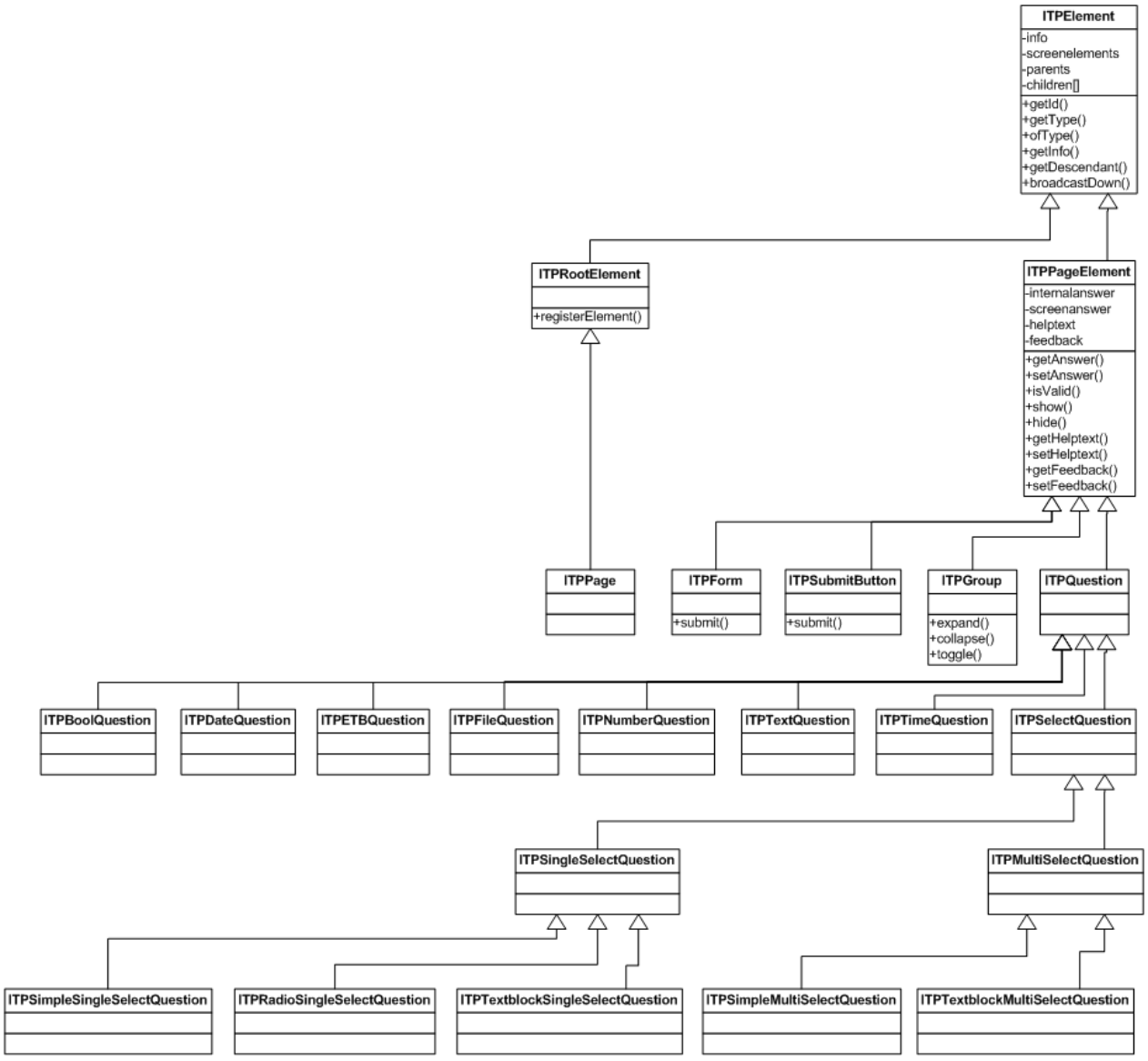
JavaScript

The JavaScript library has an open and well-defined structure. This structure can be extended as part of customization, such as to change the behavior of one type of control.

Object model

The KCM ComposerUI web form consists of a hierarchy of groups and questions, presented inside the HTML form on the HTML page. Therefore, the KCM ComposerUI web form can be modeled as a tree of objects, each representing a node in the hierarchy mentioned before. The root node of this tree represents the HTML page as a whole, whereas each sub node represents one element on the page, such as the HTML form, a group, a question or a Submit button. There can be many types of questions, such as a text question, a number question, a date question or a select question. Select questions may allow for a single or for multiple items to be selected and may or may not present text blocks as selectable items.

The JavaScript object model defines a class for each of the node types described above. The corresponding class hierarchy is shown below. The details of each class are described in Appendix E. The most important features are described in the following subsections.



ITPElement

ITPElement represents a general element in the tree. Its element type is different for each descendant of ITPElement.

ITPElement holds an info structure that basically is a JSON collection of information about the particular element. The structure and contents of this collection in a trivial manner correspond to the node sets that are passed back and forth between the templates in the XSLT. See Passing information and Appendix C for more details.

ITPElement is related to the hierarchical relations between the objects. It contains a reference to a parent object and to an array of children. It is used to transfer messages through the system and to find an element in its subtree.

ITPElement defines the general initialization of an element, which may be asynchronous. The most important part of initialization is the construction of a map of screen elements. At this point the JavaScript may manipulate the content of the screen.

ITPRootElement and ITPPage

ITPRootElement represents the root of the object tree. It offers a method to register an information structure during construction of the HTML page and implements initialization, such that:

- It constructs an object for each registered info structure
- It constructs the node tree by applying parent-child relations
- It makes sure that each object initializes itself
- It makes sure that all objects synchronize by sending messages

ITPPage derives from ITPRootElement, adding one simple function `isContentWizard`, which indicates whether or not the page shows a Content Wizard.

ITPPageElement

ITPPageElement represents a sub node in the tree, such as an element on the page. Each page element may have an answer. ITPPageElement distinguishes between an internally stored answer and a screen answer that may be visible on the page, such as the answer as it is typed in an input element. It defines answer validation, which may result in three possible values:

- Answer is correct.
- Answer cannot be accepted upon form submission, but may be shown on the screen. The user is presented with a visual indication that the answer is not valid.
- Answer cannot be accepted at any time. The previous acceptable answer is shown on the screen.

An ITPPageElement is associated with help text that provides an additional information and a feedback text that provides error messages.

ITPPageElement descendants

ITPForm and ITPSubmitButton respectively represent the HTML form and the buttons that submit this form. They both add a Submit function that allows user to click the Submit button for sending the form and passing information.

ITPGroup represents a group on the page. It implements behavior to expand, collapse, and toggle groups. The latter is implemented by processing messages that are sent by question elements.

ITPQuestion represents a question on the form. Its descendants define the specific behavior for different question types. ITPSelectQuestion and its descendants define the behavior for select questions; ITPSingleSelectQuestion for single selects and ITPMultiSelectQuestion for multi selects.

Object tree and messages

With the help of the class ITPMessage, ITPElement implements a messaging mechanism in the tree. As a result, elements in the tree can communicate without knowing each other. An element may send a message by calling the function bubbleUp. This sends the message to the parent element, so that it ends

up at the root node of the tree. From this point the message is broadcasted down the tree, allowing each node to react to and modify the message. The latter may involve attaching of an answer, or marking the message as shouldStop in which case broadcasting is stopped.

For instance, this mechanism is used to notify other nodes of a change in an answer. The class ITPGroup modifies the visibility of a group if there is a relevant change to the answer of its corresponding toggle question.

Initialization

ITPElement defines the general initialization of an element in the tree. ITPRootElement defines the initialization of the tree as a whole. It is important to be aware of the steps that are part of this process:

- The HTML page contains one global instance itppage of the class ITPPage.
- In this instance, the JavaScript in the HTML page calls registerElement for each relevant element on the page, passing the corresponding information structure.
- The onload event of the page calls itppage.initialise.
- During initialization, an object is constructed for each registered information structure. This is done by calling getITPElement on the global object itpelementfactory. This factory provides an abstraction layer to the actual construction of the objects and thus provides the basis for the extensibility model described below.
- As soon as all objects are constructed, the tree should be constructed from them by applying parent-child relationships.
- After this, all objects are initialized by calling their initialize methods. Initialization of an object may involve asynchrony. As part of the initialization of an object, its initScreenElements is called. This is the point where the object may manipulate the contents of the page. The call should return a map of references to relevant screen elements that can be used by other methods later on.
- As soon as all objects have reported back, synchronization is invoked upon each element in the tree. This induces the first traffic of messages through the tree.

Form submission

A Submit button is clicked and the corresponding object ITPSubmitButton sends a message notification about the event. This message is picked up by an instance of the class ITPForm, which then starts the process of submitting the form. This involves a call to the function prepareForSubmission, which is defined by ITPPageElement for each child of the element ITPForm. Once the preparation of a child is completed, it reports this by calling the function readyForSubmission on its parent, passing the validity of its answer. This mechanism therefore allows for asynchrony, which is for instance, used by the class ITPETBQuestion.

The object ITPForm simply submits the HTML form if all answers are valid. It is not responsible for sending any of the answers of its children. During preparation, an element should ensure that its answer is a part of the HTML form that is submitted, such as through a hidden element on the page.

jQuery

The JavaScript library uses jQuery to manipulate the page. jQuery is widely used and recommended by organizations such as Google and Microsoft. It introduces a flexible and browser independent way of querying and manipulating elements on the page and adds powerful features to the end user experience.

See www.jquery.com and www.jqueryui.com for more information.

Extensibility: an example

The extensibility of the object model allows simple and maintainable JavaScript customization of the KCM ComposerUI web forms. The following section gives an example of how the extensibility can be used. In this example, several questions are represented with thousands separators.

ITPElementFactory

First, it is important to understand the way the class ITPElementFactory, such as the global variable itpelementfactory, constructs an object from a given info structure. The XSLT that produces the new output ensures that each info structure has an attribute element type. Currently, there are 16 element types, one for each leaf in the class hierarchy.

Element type	Description
page	The main page
form	The main form
group	A group
submitbutton	A submit button
question_text	A text question
question_number	A numerical question
question_bool	A check box question
question_date	A date question
question_time	A time question
question_file	A file question
question_etbq	An editable text block question
question_ertb	An editable rich text block question
question_simplesingleselect	A single select question (no text blocks, no radio buttons)
question_radiosingleselect	A single select question (no text blocks, radio buttons)
question_simplemultiselect	A multi select question (no text blocks)
question_textblocksingleselect	A single select question (text blocks)
question_textblockmultiselect	A multi select question (text blocks)

Each class in the class hierarchy has an associated object type. Each class is registered with the global variable itpelementfactory by calling its function registerClass. Also, mappings between object types and element types can be registered, by calling the function registerMapping on itpelementfactory. For the general library, this is done in the itpelementfactorymappings source. The combination of the two types of registration provides sufficient information for the class ITPElementFactory to construct an object from an info structure.

Steps

The following three simple steps describe how to implement custom behavior:

- Implement a new class that descends from ITPElement or one of its descendants, most likely ITPQuestion.
- Make sure that the source implementing this class also registers the mapping between the class type and the proper element type.
- Redefine the xslt template producePageScript to include the newly created JavaScript source. Make sure that this source is the last one to be included. This is the only way in which the registration can take precedence over any other registration for the element type.

Example

Part of the installation of KCM ComposerUI is a custom2.zip file. This file contains an example extension to the KCM ComposerUI application. This extension serves to illustrate JavaScript customization. The customization in the custom2 application adds thousands separators to numerical questions.

The application extension custom2 is not installed by default. To install it, follow these steps:

1. Go to the configuration page of your KCM ComposerUI installation. See the chapter "Configuration" in the *KCM ComposerUI for ASP.NET and J2EE Developer's Guide*.
2. In the field Application Name, fill out "custom2" and click Submit.
3. In the OnLine application folder, a subfolder custom2 is created. Copy the contents of the newsample application to this location or unzip the newsample.zip file.
4. Unzip the contents of the custom2.zip file to the subfolder. Make sure the unzip process does not introduce an extra level custom2, subfolders `js` and `xslt` should occur immediately below the existing folder custom2.
5. For KCM ComposerUI ASP.NET only, go back to the configuration page and click the Deploy link next to the application custom2.

The application custom2 is an addition to the newsample application. It adds the following files:

- *\js\customnumberquestion.js.* The custom JavaScript file that implements the custom behavior.
- *\js\jquery.caret.min.js.* A jQuery widget that supports caret manipulation.
- *\xslt\custom\page.xsl.* An XSL file that includes the additional JavaScripts in the page.

producePageScript

The custom JavaScript file is included in the output as follows:

```
<xsl:template name="producePageScript">
  <xsl:param name="interactinfo"/>
```

```
  <xsl:param name="custom"/>
  <xsl:call-template name="producePageScript_default">
    <xsl:with-param name="interactinfo" select="$interactinfo"/>
    <xsl:with-param name="custom" select="$custom"/>
  </xsl:call-template>
  <script type="text/javascript" src="js/jquery.caret.min.js">
    <xsl:value-of select="$empty"/>
  </script>
  <script type="text/javascript" src="js/customnumberquestion.js">
    <xsl:value-of select="$empty"/>
  </script>
  </xsl:template>
```

It produces the standard JavaScript by calling the default template producePageScript and adds two script includes. This order is important, because otherwise the registration of the new class with the `itpelementfactory` would not prevail.

CustomNumberQuestion

The file customnumberquestion.js implements the class CustomNumberQuestion, which descends from ITPQuestion. It registers this class with the `itpelementfactory` and also registers a mapping between the element type question_number and the class type ITPElement.ITPPageElement.ITPQuestion.CustomNumberQuestion.

CustomNumberQuestion adds four methods: thousands, countseparators, internal2screen and screen2internal, which implement the thousands separator logic. The details of this implementation are not relevant to this example.

More importantly, CustomNumberQuestion overrides the following methods:

- **initScreenElements**: This method builds up a map of screen elements, adding to the list that is produced by the base class ITPQuestion. It finds the standard input element that is produced by the XSLT and hides it. Also, the method adds a new input, binding thousands separator logic to it.
- **setInternalAnswer**: This method ensures that the original hidden input element is synchronized with the answer provided by the end user. The content of the hidden element is posted when the form is submitted. Because this element is now guaranteed to be in sync with the answer on the screen, no additional work is required in the method prepareForSubmission.
- **getScreenAnswer**: This method returns the answer that is currently on the screen in internal format, by removing all thousands separator characters.
- **setScreenAnswer**: This method puts an answer on the screen, after adding thousands separators and the right locations.
- **validate**: This method returns whether an answer is valid and acceptable or not. It does not accept values that cannot be interpreted numerically and validates numerical values against the amount of allowed digits. If the answer is invalid, it sets a feedback text, which is presented to the end user as an error.

The combination of these overrides is sufficient to implement the custom thousands separator behavior. During initialization the custom implementation initScreenElements manipulates the screen and binds the thousands separator logic. During form entry and submission, the other overrides are combined to ensure the answer is interpreted and validated correctly.

Appendix A

XSLT templates

Templates defined in the Navigation layer (navigation.xsl)

Match	Mode	Parameters	Description
itp:interact		custom	Calls producePage to produce the page content.
itp:interact	group	custom	Calls produceGroup to produce the top level group.
itp:group		custom	Calls produceGroup to produce a child group.
itp:table		custom	Calls produceTable to produce a tabular structure.
itp:row		custom	Calls produceTableRow to produce a row in an itp:table.
itp:cell		custom	Calls produceTableCell to produce a cell in an itp:table.
itp:question		custom	• Calls produceSimpleText to produce static text. • Calls produceQuestion to produce normal questions.
itp:question	keyselection	custom	Calls produceKeyList to produce a key selection list.
xforms:submit	keyselection	custom	Calls produceKey to produce a key selection item.
xforms:item		questioninfo, custom	Calls produceOption to produce a selection option.
itp:textblock		questioninfo, custom	• Calls registerFieldSet for each field set. • Calls registerField for each field that applies to an editable text block or editable rich text block.

Match	Mode	Parameters	Description
itp:button		custom	Calls produceButton to produce a button.

Templates defined in the Customization layer (overridable.xsl)

Name	XForms context	Parameters	Description
getMaxFeatureLevel	any	-	Returns the feature level that can be handled by the transformation. Currently, the maximum level defined is 9.
producePage	itp:interact	interactinfo, custom	Produces the main page.
producePageHeader	itp:interact	interactinfo, custom	Produces the contents of the page header.
producePageTitle	itp:interact	interactinfo, custom	Produces the title in the page header.
producePageCSS	itp:interact	interactinfo, custom	Produces the css references in the page header.
producePageScript	itp:interact	interactinfo, custom	Produces the script references in the page header.
producePageBody	itp:interact	interactinfo, custom	Produces the contents of the page body.
producePageOnLoad	itp:interact	interactinfo, custom	Produces the contents of the onload attribute of the page body.
produceForm	itp:interact	interactinfo, custom	Produces the html form.
produceFormOn Submit	itp:interact	interactinfo, custom	Produces the javascript code to be executed upon form submission.
produceFormBody	itp:interact	interactinfo, custom	Produces the contents of the html form.
produceGroup	itp:group	groupinfo, custom	Produces a group.
produceGroup Header	itp:group	groupinfo, custom	Produces the header of a group.
produceGroupBody	itp:group	groupinfo, custom	Produces the contents of a group
produceTable	itp:table	tableinfo, custom	Produces a tabular structure.
produceTableRow	itp:row	rowinfo, custom	Produces a row in a tabular structure.
produceTableCell	itp:cell	cellinfo, custom	Produces a cell in a tabular structure.

Name	XForms context	Parameters	Description
produceKeyList	itp:question	keylistinfo, custom	Produces a list of key selection entries.
produceKey	xforms:submit	keyinfo, custom	Produces a key selection entry.
produceSimpleText	itp:question	textinfo, custom	Produces a bit of static text.
produceQuestion Structure	itp:question	questioninfo, custom, label, input, structuretype	Produces the general HTML structure of a question, given the specific content of the question label and the question input controls. May produce different output, based on the structuretype, which by default may either contains the value "simple" or "extended".
produceQuestion	itp:question	questioninfo, custom	Produces the content of a question. Transfers to either of the more specific question templates, possibly using the produceQuestionStructure template.
produceETBQuestion	itp:question	questioninfo, custom	Produces an editable text block question.
produceERTBQuestion	itp:question	questioninfo, custom	Produces an editable rich text block question.
produceDate Question	itp:question	questioninfo, custom	Produces a date question.
produceTime Question	itp:question	questioninfo, custom	Produces a time question.
produceText Question	itp:question	questioninfo, custom	Produces a text question.
produceNumber Question	itp:question	questioninfo, custom	Produces a number question.
produceFileQuestion	itp:question	questioninfo, custom	Produces a file question.
produceBool Question	itp:question	questioninfo, custom	Produces a check box question.
produceTextBlock MultiSelectQuestion	itp:question	questioninfo, custom	Produces a multi-select question for text blocks.
produceSimpleMulti SelectQuestion	itp:question	questioninfo, custom	Produces a multi-select question.
produceTextBlock SingleSelect Question	itp:question	questioninfo, custom	Produces a single select question for text blocks.
produceRadioSingle SelectQuestion	itp:question	questioninfo, custom	Produces a single select question rendered as radio buttons.

Name	XForms context	Parameters	Description
produceSimple SingleSelect Question	itp:question	questioninfo, custom	Produces a single select question.
produceOption	xforms:item	questioninfo, optioninfo, custom	Produces an option in a select question.
registerFieldSet	itp:fieldset	fieldsetinfo, custom	Registers the availability of a field set for an editable text block question or editable rich text block question.
registerField	itp:field	fieldinfo, custom	Registers the existence of a field inside a field set.
produceButtons	itp:interact	interactinfo, custom	Produces all form submission buttons.
produceButton	itp:button	buttoninfo, custom	Produces a form submission button.
produceButtonLabel	itp:button	buttoninfo, custom	Produces the label to be displayed on a form submission button.

Templates defined in the Support Library (support.xsl)

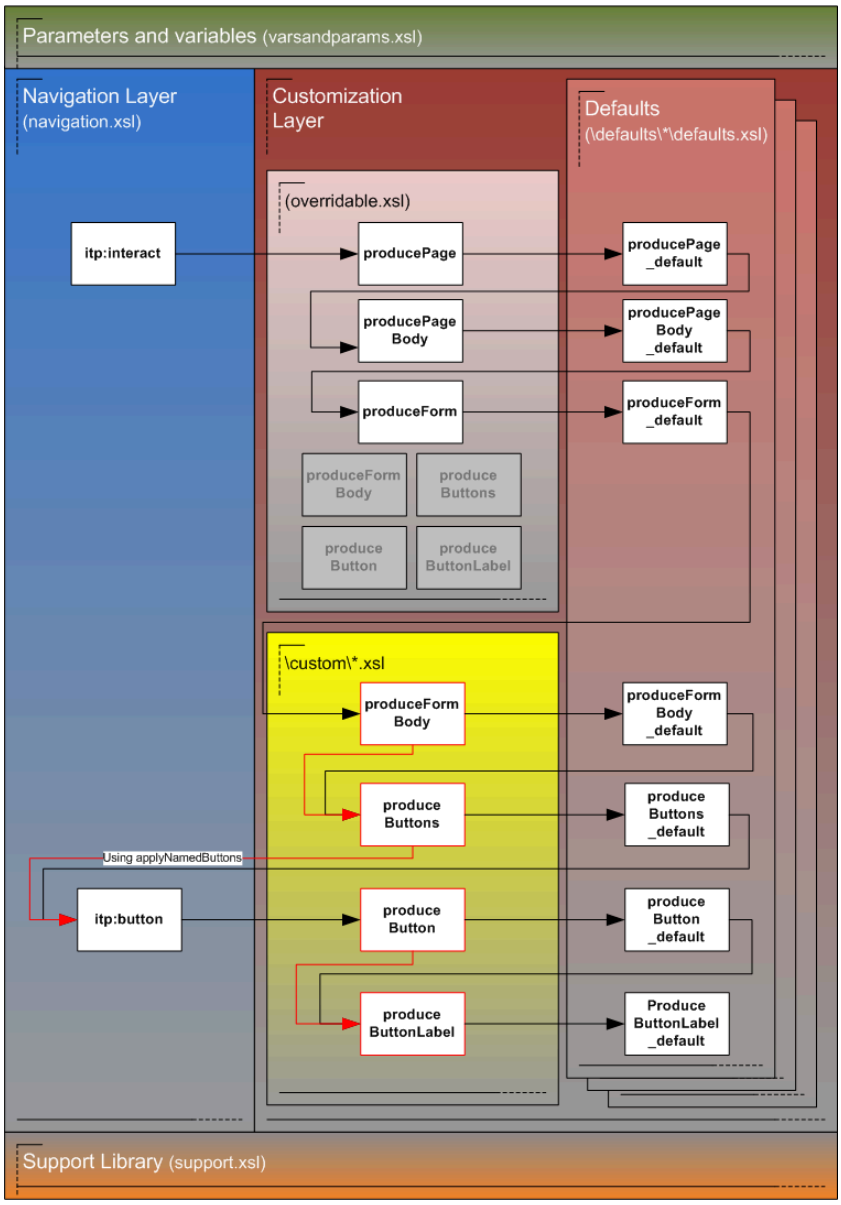
Name	XForms context	Parameters	Description
replace	any	source, tobereplaced, replacewith	Replaces all occurrences of $tobereplaced in $source by $replacewith.
escapejavavar	any	javavar	Escapes a text value for use in Javascript string values.
escapeurlparam	any	urlparam	Escapes a text value for use in a query string.
buildTextBlock PreviewUrl	any	sessionid, rprj, rusr, srv, port, env	Creates a URL to the text block preview. Appending a text block identifier to this URL results in a URL that shows the preview for the corresponding text block.
applySelected Options	itp:question	questioninfo, custom, enforceordering	Drills down to the xforms:item level, only for those items that should be selected by default, possibly enforcing the order in which the items are processed to follow the order in the default.

Name	XForms context	Parameters	Description
applyUnselected Options	itp:question	questioninfo, custom	Does the same for the items that should not be selected by default.
applyAllOptions	itp:question	questioninfo, custom	Does the same for all items, namely, ignoring the default.
applyNamedButtons	itp:question	names, custom	Drills down to the itp:button level, only for the buttons with a submission attribute named in the $names parameter (space separated list).
applyOtherButtons	itp:question	exclude_names, custom	Does the same for the buttons not in the $exclude_names parameter.
buildInteractInfo	itp:interact	-	Builds a node set containing interact info.
buildGroupInfo	itp:group	-	Builds a node set containing group info.
buildTableInfo	itp:table	-	Builds a node set containing table info.
buildTableRowInfo	itp:row	-	Builds a node set containing row info.
buildTableCellInfo	itp:cell	-	Builds a node set containing cell info.
buildTextInfo	itp:question	-	Builds a node set containing text info.
buildQuestionInfo	itp:question	-	Builds a node set containing question info.
buildKeyListInfo	itp:question	-	Builds a node set containing keylist info.
buildKeyInfo	xforms:submit	-	Builds a node set containing key info.
buildOptionInfo	xforms:item	questioninfo	Builds a node set containing option info.
buildFieldSetInfo	itp:fieldset	questioninfo	Builds a node set containing field set info.
buildFieldInfo	itp:field	questioninfo	Builds a node set containing field info.
buildButtonInfo	itp:button	-	Builds a node set containing button info.

Name	XForms context	Parameters	Description
buildSuspendButton Info	any	-	Builds a node set containing button info for the suspend button.

Appendix B

Example flow



An example illustrating the control flow between the different layers with the customization of the Example.

Appendix C

XSLT info structures

InteractInfo

Parameter	Description
id	A unique identifier for the form.
type	This attribute indicates the type of form that is presented. Currently defined values: • **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.
subtype	This attribute distinguishes between different forms with type="content-wizard". Currently defined values: • **main**: The main Content Wizard form. • **ETB**: The subsequent form, presenting editable rich text blocks.
lang	The language currently used by KCM to generate dates, numbers and other language-dependent output.
guilang	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 and "CC" is the ISO-3166 country code. For example: • The KCM language "ENG" (English localized for the UK) maps to "en-GB." • The KCM language "NLB" (Dutch localized for Belgium) maps to "nl-BE."
version	The version of KCM that generated the interact.xml file.

Parameter	Description
featurelevel	Indicates the FORM features that actually occur in the form. Currently defined levels: 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, RECORDSET, SHOW/SHOWNOT. 5 TEXT_BLOCK questions. 6 Support for DATE selection. 7 Advanced GROUP TOGGLE behavior. 8 GROUP TOGGLE behavior with multiple conditions. Each level includes the features of the previous levels.
haserrors	This attribute's value "true" indicates that the form is a resend 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".
title	The title of the form as defined on the FORM statement or in the Form editor.
source	Optional attribute indicating the context in which the transformation is executed. In the context of KCM ComposerUI, this attribute is empty. If the transformation is used for the preview of the Form Editor, the value will be "formeditor".

GroupInfo

Parameter	Description
id	Sequence number that uniquely identifies this group within the form.
title	The heading for the group.
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 an collapsible/expandable element.
expanded	Indicates the original state of the group. If the value is true an interactive client should render this group expanded.
togglesource	Indicates the question that controls whether or not this group should be toggled visible/hidden.

Parameter	Description
togglevalue	Indicates the value on which the group is shown or hidden.
togglecondition	Indicates the condition on which the group is shown or hidden. Possible conditions: • "=" Shows the group only if the current value of the toggle-source question matches the togglevalue. • "<>" Shows the group only if the current value of the toggle-source question does not match the togglevalue. • "<" Shows the group only if the current value of the toggle-source question is smaller than the togglevalue (numeric questions only). • ">" Shows the group only if the current value of the toggle-source question is larger than the togglevalue (numeric questions only). • "<=" Shows the group only if the current value of the toggle-source question is smaller than or equal to the togglevalue (numeric questions only). • ">=" Shows the group only if the current value of the toggle-source question is larger than or equal to the togglevalue (numeric questions only). • "CONTAINS" Shows the group only if the current value of the toggle-source question contains the togglevalue (select questions only). • " !CONTAINS" Shows the group only if the current value of the toggle-source question does not contain the togglevalue (select questions only). • " IN" Shows the group only if the current value of the toggle-source question equals one of the options in the togglevalue. • " !IN" Shows the group only if the current value of the toggle-source question equals none of the options in the togglevalue.
parentgroup	Identifier of the parent group.

TableInfo

Parameter	Description
id	Label that uniquely identifies this table within the form.
row	Declaration of the number of rows in the table.
columns	Declaration of the number of columns in the table.
label	The heading of the table.

RowInfo

Parameter	Description
id	Label that uniquely identifies this row within the table.
row	Declaration of the columns number in the table.

CellInfo

No content defined for table cell info, yet.

TextInfo

Parameter	Description
text	The static text to be produced.
renderingcontext	Whether or not the text is produced inside a table ("table" or "standard").

QuestionInfo

Parameter	Description	
id	A unique identifier of the question, determined by the ID (…) keyword in the KCM Master Template, or the identifier on the advanced tab of the Form editor.	
idhash	An unique identifier of the question that is suitable for use as id attribute on html tags.	
ref	A unique reference identifier that is used to correlate information within the XForms.	
default	The default answer to the question.	
typename	The answer type, which can have the following values:	
	etbq	Editable text block.
	ertbq	Editable rich text block.
	bool	Boolean.
	date	Date.
	time	Time.
	text	Text.
	number	Number.

Parameter	Description	
	file	File.
	The controltype gives more detailed information about the question type.	
length	The maximum length of a text answer.	
totaldigits	The maximum total amount of digits of a number answer.	
fractiondigits	The maximum amount of fractional digits of a number answer.	
paragraphset	Indicates the view that should be used to select text blocks from.	
textblockserver	A node with information on the KCM Repository Server that can serve text block previews. The buildTextBlockPreviewUrl template in the Support Library abstracts from the intricacies of this information.	
server	The name of the server.	
port	The port on which the server can be accessed.	
environment	The environment to be used when retrieving a text block preview.	
repositoryuser	The repository user providing the context from which text block revisions should be retrieved.	
repositorystatus	The status, such as development or published, determining the text block revisions to be retrieved.	
repositoryproject	The project from which text bock revisions should be retrieved.	
orderresponse	Whether or not the end user should be able to order the selected options of a multi-select question.	
readonly	If this element is part of the QuestionInfo structure, the question should be presented for read-only.	
helptext	The help text providing extra information, to be presented along with the question.	
feedback	A message providing an error or warning, to be presented along with the question.	
feedbackreason	The nature of the feedback message.	
nrofoptions	The number of options of a select question.	
renderingcontext	Whether or not the text is produced inside a table ("table" or "standard").	
controltype	The type of the question control, which can have the following values:	
	etbq	Editable text block
	ertbq	Editable rich text block
	bool	Boolean
	date	Date
	time	Time
	text	Text
	number	Number
	file	File
textblockmultiselect	Text block multi-select.	

Parameter	Description
simplemultiselect	Non-text block multi-select.
textblocksingleselect	Text block single select.
radiosingleselect	Non-text block radio button single select.
simplesingleselect	Non-text block single select.
parentgroup	Identifier of the parent group.
minselect	The minimum number of answers to select for a select question.
maxselect	The maximum number of answers to select for a select question.

KeyListInfo

Parameter	Description
keylistprompt	Description of the key selection list.

KeyInfo

Parameter	Description
submission	The value to be submitted as "submission" form field when the key is selected.
label	Short representation of the record in the key selection list.
screenfields	Human readable representation of a record in the key selection list.
rank	The rank number of the record within the key selection list.

OptionInfo

Parameter	Description
label	Human readable representation of the option. In the case of a text block option, this parameter contains both the name of the text block and its description.
tbklabel	The name of the text block for this option (if any).
tbkdescription	The description of the text block for this option (if any).
value	The value to be submitted if the option is selected upon form submission.

Parameter	Description
previewvalue	The value (if any) to be passed to the text block preview page.
defaultselected	The OptionInfo should initially be selected if its structure contains this parameter.
order	An optional number determining the ordering of the options within the selection.

FieldsetInfo

Parameter	Description
textblockid	An identifier of the editable rich text block to which the field set applies.
fieldsetname	The name of the field set.

FieldInfo

Parameter	Description
textblockid	An identifier of the editable rich text block to which the field set containing the field applies.
fieldsetname	The name of the field set that contains the field.
fieldname	The name of the field.

ButtonInfo

Parameter	Description
id	A unique identifier for the button in the form.
label	Human readable representation to be presented on the button.
submission	The value to be submitted as "submission" form field when the button is pressed.

Appendix D

XHTML

This Appendix presents a drill-down into the XHTML output as it is, before it is manipulated by JavaScript. Items between [brackets] refer to variables or variable content.

[PAGE]

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
  <head>
    <title>[PAGETITLE]</title>
    <link rel="stylesheet" type="text/css" href="cs/ui/jquery-ui-1.7.2.custom.css" />
    <link rel="stylesheet" type="text/css" href="cs/div_oriented.css" />
    <link rel="stylesheet" type="text/css" href="cs/ui.itppreview.css" />
    <link rel="stylesheet" type="text/css" href="cs/ui.itpselect.css" />
    <script type="text/javascript" src="js/jquery-1.4.1.min.js"></script>
    <script type="text/javascript" src="js/jquery-ui-1.7.2.custom.min.js"></script>
    <script type="text/javascript" src="js/ui.itppreview.js"></script>
    <script type="text/javascript" src="js/ui.itpselect.js"></script>
    <script type="text/javascript" src="js/itpcommon.js"></script>
    <script type="text/javascript" src="resources/[LANGUAGE].js"></script>
    <script type="text/javascript" src="support/javascript/itp_ajax.js"></script>
    <script type="text/javascript" src="support/javascript/itp_textblocks.js"></script>
    <script type="text/javascript" src="support/javascript/mktree.js"></script>
    <script type="text/javascript" src="support/javascript/parse.js"></script>
    <script type="text/javascript" src="js/itpmessage.js"></script>
    <script type="text/javascript" src="js/itpelementfactory.js"></script>
    <script type="text/javascript" src="js/itpelementfactorymappings.js"></script>
    <script type="text/javascript" src="js/itpelement.js"></script>
    <script type="text/javascript" src="js/itprootelement.js"></script>
    <script type="text/javascript" src="js/itppage.js"></script>
    <script type="text/javascript" src="js/itppageelement.js"></script>
    <script type="text/javascript" src="js/itpform.js"></script>
    <script type="text/javascript" src="js/itpgroup.js"></script>
    <script type="text/javascript" src="js/itpsubmitbutton.js"></script>
    <script type="text/javascript" src="js/itpquestion.js"></script>
    <script type="text/javascript" src="js/itpnumberquestion.js"></script>
    <script type="text/javascript" src="js/itpboolquestion.js"></script>
    <script type="text/javascript" src="js/itpdatequestion.js"></script>
    <script type="text/javascript" src="js/itptimequestion.js"></script>
    <script type="text/javascript" src="js/itpsimplesingleselectquestion.js"></script>
    <script type="text/javascript" src="js/itpradiosingleselectquestion.js"></script>
    <script type="text/javascript" src="js/itpsimplemultiselectquestion.js"></script>
    <script type="text/javascript" src="js/itptextblocksingleselectquestion.js"></
script>
    <script type="text/javascript" src="js/itptextblockmultiselectquestion.js"></
script>
    <script type="text/javascript" src="js/itpetbquestion.js"></script>
    <script type="text/javascript" src="js/itpfilequestion.js"></script>
```

```
    <script type="text/javascript" src="js/itptextquestion.js"></script>
    <script type="text/javascript">
      var implementation ='[IMPLEMENTATION]';
      var uploadpath='[UPLOADPATH]';
      var amInSecureMode=true/false;
      var itppage = itpelementfactory.getITPElement([JSONINFO]);</xsl:text>
    </script>
  </head>
  <body onload="itppage.initialise(function(pObj){});">
    <div class="main [TYPE]">
      <script type="text/javascript">
        itppage.registerElement([JSONINFO]);
      </script>
      <form id="id_form" action="[LINKPAGE]" enctype="multipart/form-data" accept-
charset="UTF-8" method="post">
[GROUP]n
        <div class="buttons">
[BUTTON(ok back1 cancel)]n
[BUTTON(other)]n
        </div>
      </form>
    </div>
    <input class="initmarker" style="display: none;" type="checkbox"/>
  </body>
</html>
```

[GROUP]

```
<script type="text/javascript">
    itppage.registerElement([JSONINFO]);
  </script>
  <div class="group level[LEVEL]" id="id_[ID]_container">
    <fieldset>
      <legend>
        <span class="groupheader" id="id_[ID]_groupheader">
          <span class="grouptitle">
            [TITLE]
          </span>
        </span>
      </legend>
      <div class="groupbody" id="id_[ID]_groupbody">
[GROUP/TABLE/QUESTION]n
      </div>
    </fieldset>
  </div>
```

[TABLE]

```
  <table class="itptable">
[ROW]n
  </table>
```

[ROW]

```
<tr>
[CELL]n
  </tr>
```

[CELL]

```
<td>
[QUESTION]n
  </td>
```

[KEYLIST]

```
<div class="keylist group level[LEVEL]" id="id_keylist_container">
    <fieldset title="[KEYLISTPROMPT]">
      <legend>
        <span class="groupheader" id="id_keylist_groupheader">
          <span class="grouptitle">
            [KEYLISTPROMPT]
          </span>
        </span>
      </legend>
      <div class="groupbody" id="id_[ID]_groupbody">
[KEY]n
      </div>
    </fieldset>
  </div>
```

[KEY]

```
  <script type="text/javascript">
    itppage.registerElement([JSONINFO]);
  </script>
  <div class="key" id="id_[ID]_container">
    <input type="submit" id="id_[ID]_submit" name="submission" value="[SUBMISSION]"/>
  </div>
```

[SIMPLETEXT]

```
<div class="statictext">
    [TEXT]
  </div>
```

[QUESTION]

```
<!-- Note: in a Content Wizard some section questions are suppressed. -->
  <script type="text/javascript">
    itppage.registerElement([JSONINFO]);
  </script>
  <div class="question [STRUCTURETYPE] [CONTROLTYPE] [RENDERINGCONTEXT] [PARITY]
 question_[PARITY]" id="id_[IDHASH]_container">
    <div class="label [STRUCTURETYPE] [isempty/haslabel]">
      <span class="annotation">
        <img id="[ID]_help" src="cs/img/helpicon.gif" class="help" alt="help"
 title="[HELPTEXT]" [style="visibility:hidden"]/>
        <img id="[ID]_feedback" src="cs/img/feedbackicon.gif" class="feedback"
 alt="feedback" title="[FEEDBACK]" [style="visibility:hidden"]/>
      </span>
      <span class="labelwrap">
        <label for="[REF]">
          [LABEL]
        </label>
      </span>
    </span>
    <div class="input [STRUCTURETYPE]">
[ETBQ/DATE/TIME/TEXT/NUMBER/FILE/BOOL/TEXTBLOCKMULTISELECT/SIMPLEMULTISELECT/
TEXTBLOCKSINGLESELECT/RADIOSINGLESELECT/SIMPLESINGLESELECT]
    </div>
  </div>
```

[ETBQ]

```
<script language="javascript">
    online_tbk_manager.registerTextBlock('[REF]', [READONLY?]);
   [online_tbk_manager.registerFieldSet('[REF]', '[FIELDSETNAME]');
     [online_tbk_manager.registerField('[REF]', '[FIELDSETNAME]', '[FIELDNAME]);]n
   ]n
  </script>
  <input class="ui-widget" type="text" id="id_[REF]" name="[REF]"
 value="[DEFAULT]" [disabled="true"]>
  </input>
```

[DATE]

```
  <input class="ui-widget" type="text" id="id_[REF]" name="[REF]" size="10"
 maxlength="10" value="[DEFAULT]" [disabled="true"]/>
```

[TIME]

```
<input class="ui-widget" type="text" id="id_[REF]" name="[REF]" size="8" maxlength="8"
 value="[DEFAULT]" [disabled="true"]/>
```

[TEXT length >=50]

```
  <textarea id="[REF]" name="id_[REF]" cols="40" cols="[10 or less]" [disabled="true"]>
    [DEFAULT]
  </textarea>
```

[TEXT length <50]

```
<input class="ui-widget" type="text" id="id_[REF]" name="[REF]"
 value="[DEFAULT]" [maxlength="[LENGTH]" size="[LENGTH]"] [disabled="true"]/>
```

[NUMBER]

```
<input class="ui-widget" type="text" id="id_[REF]" name="[REF]" size="[SIZE]"
 maxlength="[SIZE]" value="[DEFAULT]" [disabled="true"]/>
```

[FILE]

```
<input class="ui-widget" type="file" id="id_[REF]" name="[REF]"
 size="40" [disabled="true"]/>
```

[BOOL]

```
<input class="ui-widget" type="checkbox" id="id_[REF]" name="[REF]" [checked="Y"]
 [disabled="true"]/>
```

[TEXTBLOCKMULTISELECT]

```
<select id="[REF]" name="id_[REF]" size="[NROFOPTIONS]"
 multiple="multiple" [disabled="true"]>
[OPTION]n
    </select>
```

[SIMPLEMULTISELECT]

```
<select id="[REF]" name="id_[REF]" size="[NROFOPTIONS]"
 multiple="multiple" [disabled="true"]>
[OPTION]n
    </select>
```

[TEXTBLOCKSINGLESELECT]

```
<select id="[REF]" name="id_[REF]" size="[NROFOPTIONS]" [disabled="true"]>
[OPTION]n
    </select>
```

[RADIOSINGLESELECT]

```
<select id="[REF]" name="[REF]" size="[NROFOPTIONS]" [disabled="true"]>
[OPTION]n
    </select>
```

[SIMPLESINGLESELECT]

```
<select id="[REF]" name="[REF]" size="[NROFOPTIONS]" [disabled="true"]>
[OPTION]n
    </select>
```

[OPTION]

```
<option value="{$optioninfo/value}" [selected="true"]>
[LABEL]
  </option>
```

[BUTTON]

```
<script type="text/javascript">
    itppage.registerElement([JSONINFO]);
  </script>
  <span id="id_[ID]_container">
    <input type="submit" id="id_[ID]_submit" name="submission" value="[SUBMISSION]"/>
  </span>
```

Appendix E

JavaScript library

ITPElement

An ITPElement represents a node in the tree, which knows both its parent and its children. This tree has a unique ID and holds a class type, an info structure and a collection of associated screen elements. It supports asynchronous initialization and can deal with messages that appear and are broadcasted down the tree.

Constructor	
ITPElement(pInfo)	Simple initialization of local variables.
Public methods	
getId()	Returns the elementid from the info structure.
getType()	Returns the class type. Class types should reflect the class hierarchy, by listing all ancestor classes, separated by a dot, such as ITPElement.ITPRootElement.ITPPage.
ofType(pType)	Returns whether the object is of the class type pType, or of a descendant type.
getInfo()	Returns the info structure.
initialize(pCallback, pSuppressImmediateCallback)	Initializes the object by calling initScreenElements, registering pCallback as the callback method. For ITPElement, which does not have any asynchronous initialization, the callback method is called immediately, unless pSuppressImmediateCallback equals true.
Other methods	
synchronize()	Triggers sending of all messages that may be relevant to other nodes in the tree. To be overridden in descendant classes.
registerParent(pElement)	Associates the node with parent node pElement, which should be a descendant of ITPElement.
registerChild(pElement)	Associates the node with child node pElement, which should be a descendant of ITPElement.
getDescendant(pId, pRequiredType)	Returns a node from the subtrees defined by the children of the current node if it has the object id equal to pId and if it is of type pRequiredType. Returns null if such a node cannot be found.

bubbleUp(pMessage)	Passes the message pMessage on to the parent of the current node, if the current node has been associated with a parent and allowBubbleUp(pMessage) returns true. Calls broadcastDown(pMessage) otherwise.
broadcastDown(pMessage)	Calls processMessage(pMessage), before calling broadcastDown(pMessage) for each of the associated children. Note that processMessage may tag the message to "shouldStop()", in which case the broadcasting will be aborted.
allowBubbleUp(pMessage)	Returns whether the message is allowed to be displayed. Can be used to limit the range of a message to a subtree. To be overridden in descendant classes.
processMessage(pMessage)	Is called for each message that is broadcasted through the node. Should return pMessage, possibly after modifying it. Can be used to respond to messages. To be overridden in descendant classes.
initScreenElements()	Returns a collection of screen elements, which is stored in the attribute screenelements of the object. Is called as a first step in the initialization process.

ITPRootElement

Descendant of ITPElement that represents the root of the tree. It allows for the registration of JSON info structures, which is expanded as an object tree upon initialization.

Constructor	
ITPRootElement(pInfo)	Simple initialization of local variables.
Public methods	
registerElement(pInfo)	Registers an info structure, which is expanded to an ITPElement descendant during construction of the tree.
registerErrorElement(pElement)	Registers elements that report errors. Only the first element is stored. This is used to scroll the first error into view after initializing the page.
Overrides	
initialise(pCallback)	Constructs an object for each registered info structure, using the global variable `itpelementfactory`. Associates parents and children, thus implicitly constructing the tree. Initializes all objects, passing a callback function that checks whether all constructed object are initialized or not. As soon as this is the case, calls synchronize to enforce message synchronization and calls back through its own callback function.
synchronize()	Calls synchronize on each of its elements.
getDescendant(pId, pRequiredType)	Uses the map of registered elements as an index to implement a more optimal search for the descendant.

processMessage(pMessage)	Stops the broadcasting of any messages as long as initialization has not yet been completed.
Other methods	
-	-

ITPPage

Just a specific descendant of ITPRootElement.

Constructor	
ITPPage	Trivial.
Public methods	
isContentWizard()	Returns whether the current tree represents a Content Wizard form.
Overrides	
-	-
Other methods	
-	-

ITPPageElement

Descendant of ITPElement that represents a sub node in the tree. The element may be associated with an answer, consisting of two components: an internal answer and a screen answer. ITPPageElement supports the validation of an answer, which may either be valid, invalid (may not be submitted), or unacceptable (may not be on the screen). The element may be hidden. The element may be associated with a Help text and a feedback text. The element may have its own representation of the empty (null) value.

Constructor	
ITPPageElement	Simple initialization of local variables.
Public methods	
getAnswer()	Returns the current internal answer.
setAnswer(pAnswer)	Attempts to set the answer pAnswer. Returns the validation result. If this is valid, applies the answer.
isValid()	Returns the 'worst' validation result of the internal answers of the sub tree defined by the current object.
show()	Shows the element screenelements.container of the current object.
hide()	Hides the element screenelements.container of the current object.

getHelptext()	Returns the current Help text.
setHelptext(pHelptext)	Sets the current Help text to pHelptext.
getFeedback()	Returns the current feedback text.
setFeedback(pFeedback)	Set the current feedback text to pFeedback.
scrollIntoView	Scroll the current element into view.
Overrides	
initialize(pCallback, pSuppressImmediateCallback)	Sets the default answer, the Help text and the feedback text.
initScreenElements()	Finds the element with the id "[ID]_container", where [ID] is the object identifier of the object and stores this as result.container. Finds the elements within this container with the id "[ID]_help" and "[ID]_feedback" and stores them as result.helpimage and result.feedbackimage, respectively. Adds the spans result.helpspan and result.feedbackspan with the proper JQuery icon classes associated, making sure all Help and feedback indications are initially hidden. They may be shown during later steps in the initialization process.
synchronize()	Sends the current internal value of the element as part of a "valuechange" message.
Other methods	
scrollIntoView	Ensures the element is part of the visible part of the page.
prepareForSubmission(pCallback)	Method to be called before a form is submitted to allow nodes in the tree to prepare for submission, such as used by editable rich text block questions. This method is intended to be called between parents and children in the tree only. Children have to report back through the readyForSubmission method of their parent.
readyForSubmission(pElement, pValid)	Method to be called from child to parent, once a child has finished preparing for submission, passing itself as a pElement and the current validation status as pValid.
getInternalAnswer	Returns the internal answer.
setInternalAnswer(pAnswer)	Validates pAnswer and sets the internal answer to pAnswer if it is not unacceptable. Makes sure that the null value of the class is never deemed unacceptable. Synchronizes if the internal answer is changed.
applyInternalAnswer()	Ensures the internal answer is applied to the screen.
getScreenAnswer()	Returns the answer that is currently displayed on the screen. To be overridden by descendant classes.
setScreenAnswer(pAnswer)	Ensures the answer on the screen is set to pAnswer. To be overridden by descendant classes.
applyScreenAnswer()	Ensures the screen answer is set as the internal answer. Unless it is unacceptable, in which case the internal answer is applied to the screen answer.

validate(pAnswer)	Simply returns valid. To be overridden by descendant classes.
getNullValue()	Simply returns "". To be overridden by descendant classes.

ITPForm

Descendant of ITPPageElement that represents a form. Allows for programmatic submission of the form to ensure that the correct "submission" is posted.

Constructor	
ITPForm	Trivial
Public methods	
submit(pCallback)	Attempts to submit the form, which may be an asynchronous process. Calls back prior to the actual submission of the form.
Overrides	
initScreenElements()	Finds the form element by looking for an element with an id equal to the current object id. Appends a hidden "submission" input element to the form to pass the button through which the form is submitted.
processMessage	Processes "submitpressed" messages by applying their value to the hidden "submission" input element and calling submit().
Other methods	
-	-

ITPSubmitButton

Descendant of ITPPageElement that represents a submit button or a key selection button on the form. Replaces the standard input button by a button with jQuery-classes.

Constructor	
ITPSubmitButton	Trivial
Public methods	
submit()	Sends a "submitpressed" message with the submission value of the current button, to be picked up by the ITPForm node in the tree.
Overrides	

initScreenElements	Finds the input button as the only "input" child element of the container. Creates a new button element with the proper JQuery classes associated, binding the submit() method to its onclick event. If the button is a key selection button, indicated by the existence of the info.screenfields attribute, its label will be equal to info.screenfields. Otherwise it will be equal to info.label. For key selection buttons an additional class is added, indicating whether the selection button has an odd or even rank to allow for the old alternating representation.
Other methods	
-	-

ITPGroup

Descendant of ITPPageElement that represents one group on a form. It adds the functionality of expanding and collapsing of groups. It shows/hides automatically by picking up relevant messages about value changes ("group toggling"). It may be associated with a preview URL and automatically adds icons, such as expand/collapse, and preview, if necessary.

Constructor	
ITPGroup	Simple initialization of local variables.
Public methods	
expand	Ensures the group, if expandable, is expanded.
collapse	Ensures the group, if expandable, is collapsed.
toggle	Ensures the group, if expandable, changes its expand state.
Overrides	
initialise	Ensures the expand state is initialized correctly.
initScreenElements	Lifts the group representation to include group expanders, preview links.
show()	Shows the group container by folding it.
hide()	Hides the group container.
processMessage	Processes value change messages to implement group toggling. Also used to connect the group to its toggle in the case of a Content Wizard.
Other methods	
-	-

ITPQuestion

A descendant of ITPPageElement, representing a question on a form, which serves as a base class for all question types.

Constructor	
ITPQuestion	Trivial
Public methods	
-	-
Overrides	
initScreenElements()	Retrieves the label elements to be able to add error information, if required, later on.
setFeedback(pFeedback)	Writes the error text as part of the question label.
Other methods	
-	-

ITPBoolQuestion

A descendant of ITPQuestion that represents a check box question on a form. Adds a hidden check box to make sure the value FALSE is posted if the question is left unchecked. No validation required.

Constructor	
ITPBoolQuestion	Trivial
Public methods	
-	-
Overrides	
initScreenElements()	Gets the checkbox, assuming it is the only input element within the container. Ensures its value is set to "TRUE" and binds the applyScreenAnswer method to its onclick and onchange events. Adds an additional hidden checkbox with value "FALSE", to ensure a value is always posted for this question.
setInternalAnswer(pAnswer)	Converts pAnswer to "true()" or "false()", these values are also used to express toggling in the XForms, and applies it to the internal answer. Synchronizes the hidden value="FALSE" checkbox with the new value.
getScreenAnswer()	Returns "true()" if the checkbox on the screen is checked, "false()" otherwise.
setScreenAnswer(pAnswer)	Sets the checkboxes to the correct checked states, given the answer pAnswer.

other methods	
-	-

ITPDateQuestion

A descendant of ITPQuestion that represents a date question on a form. Associates a date picker control with the input field. Implements some simple date validation, to be extended and improved. Ensures all screen representation is done in a screen representation format, whereas internal representation remains in the ITP XForms format.

Constructor	
ITPDateQuestion	Trivial
Public methods	
-	-
Overrides	
initScreenElements()	• Transforms a simple input field, which is still used to post the internal answer, to the input with fixed size and associated JQuery classes. • Binds the method applyScreenAnswer to the events onkeyup and onblur. • Ensures that correct dateControlMask is used: array which determines the order showing the elements of the date. • The locale-specific masks are defined in the language specific resources \[lan].js files, such as en.js, nl.js.
setInternalAnswer(pAnswer)	Sets the internal answer, which should be in the XForms format, both in the local variable and in the hidden input control.
getScreenAnswer()	Returns the screen answer in KCM format.
setScreenAnswer(pAnswer)	Converts pAnswer from KCM to screen format and applies it to the screen. Associates a date picker with the control, if it hasn't already been done.
validate(pAnswer)	Validates a date value.
Other methods	
ITP_to_screen(pITPAnswer)	Converts a KCM format answer to screen format.
Screen_to_ITP(pScreenAnswer)	Converts a screen format answer to KCM format.
setDateAnswer(dateText, inst)	Callback function for the date picker widget.
getDateFormat(pMask, pSeparator)	Transforms an old array dateControlMask to a date format string.

ITPETBQuestion

A descendant of ITPQuestion, representing an Editable Text Block Question on a form. Adds an iframe in which the editorPage page is loaded and handles callbacks from this page. No validation required.

Constructor	
ITPETBQuestion	Trivial
Public methods	
-	-
Overrides	
initialise(pCallback, pSuppressImmediateCallback)	Calls ancestor's initialize method, suppressing callbacks.
initScreenElements()	Replaces the trivial input element by an iframe in which the editorPage is loaded. Registers the text block with the global text block manager, which is of the class ITPETBManager, defined in the same JavaScript file. Stores a reference to the window in which the editorPage is loaded.
prepareForSubmission(pCallback)	Triggers a save of the text block in the editor window, which will eventually result in a callback to setFinalValue.
Other methods	
getInitialValue()	Returns the value of the text block before editing.
setFinalValue(pValue)	Callback function for saving of a text block. Writes its post value to the hidden input element and reports readyForSubmission to its parent.
windowInitialised()	Callback function, which is called from the editorPage as soon as the TinyMCE editor has been initialized. Invokes editing of the text block in the editor.
textblockLoaded()	Callback function, which is called from the text block manager when the text block has been loaded. This means that the question is fully initialized, and the object reports that by invoking its initialization callback.

ITPERTBQuestion

A descendant of ITPQuestion that represents an Editable Rich Text Block Question on a form. It presents the question as a clickable text with an adjacent button and uses ActiveX to open KCM/Workstation to allow editing in the Rich Text Block. Validates whether the Documents are still open for editing.

Constructor	
ITPERTBQuestion	Trivial
Public methods	
-	-

Overrides	
initScreenElements()	Replaces the trivial input element by an input containing the question text and an edit button that triggers editBlock.
prepareForSubmission(pCallback)	Checks that the document is closed and then triggers saveBlock.
axEvent(pEvent)	Changes the edit button icon depending on the state (open or closed) of the rich text block document.
Other methods	
editBlock()	Uses ActiveX to activate EditDocument for the Rich Text Block document.
saveBlock()	Uses ActiveX to upload the edited rich text block document to KCM Core.

ITPFileQuestion

A descendant of ITPQuestion that represents a file question on a form. If ActiveX can be used, it transforms the file input to a text input with an adjacent button, using the ActiveX control. No validation, existence of the file checked by ActiveX upon submission.

Constructor	
ITPFileQuestion	Trivial
Public methods	
-	-
Overrides	
initScreenElements()	Turns the HTML file upload control into an ActiveX-version, if this is possible (IE) and requested (fileEditActiveX variable).
prepareForSubmission	If ActiveX is used, uploads the file to KCM Core and stores the result in a hidden input for posting.
getScreenAnswer()	If ActiveX is used, returns the selected file path. This is not possible by using the HTML control.
setScreenAnswer(pAnswer)	If ActiveX is used, it sets the selected file path. This is not possible using the HTML control.
Other methods	
browse()	Opens the file browser through the ActiveX control.
testActiveX(pScreenElements, pAttemptsLeft)	Checks if the ActiveX control is available. If this it not the case, it will retry for pAttemptsLeft times. If this fails, the question will fall back to the standard HTML file upload control.

ITPNumberQuestion

A descendant of ITPQuestion that represents a number question on a form. Transforms the simple input into separate inputs, one for decimal and one optional for fractional digits. Validates against maximum total and fraction length.

Constructor	
ITPNumberQuestion	Trivial
Public methods	
-	-
Overrides	
initScreenElements()	Replaces the simple input element by two: one for the decimal digits and one optional for the fractional digits. Binds to events and associates jQueryUI classes.
setInternalAnswer(pAnswer)	Applies pAnswer to the internal answer variable and the hidden input element.
getScreenAnswer()	Returns a representation of the values from the decimal and the fractional digit elements, separated by a dot.
setScreenAnswer(pAnswer)	Splits pAnswer based on the dot and applies it to the decimal and fractional digit elements.
validate(pAnswer)	Checks whether pAnswer is numeric and if the number of decimal and fractional digits are within the limitations set by totaldigits and fractiondigits. Note that this has always been implemented as "maximum number of decimal digits equals totaldigits minus fractiondigits."
Other methods	
-	-

ITPTextQuestion

A descendant of ITPQuestion that represents a text question on a form. No manipulation, except for event handling on input. No validation.

Constructor	
ITPTextQuestion	Trivial
Public methods	
-	-
Overrides	
initScreenElements()	Binds to onkeyup and onblur events.
getScreenAnswer()	Returns the answer in the text input.

setScreenAnswer(pAnswer)	Sets the answer in the text input to pAnswer.
Other methods	
-	-

ITPTimeQuestion

A descendant of ITPQuestion that represents a time question on a form. Transforms simple input box into two, one for the hours and one for the minutes. Only accepts hours in [0, 23] and minutes in [0, 59].

Constructor	
ITPTimeQuestion	Trivial
Public methods	
-	-
Overrides	
initScreenElements()	Replaces the simple input element by an hour and a minute input element. Binds events and associated JQuery classes.
setInternalAnswer(pAnswer)	Applies pAnswer to the internalanswer variable and to the hidden input box.
getScreenAnswer()	Returns the answer on the screen in the format hh:mm:00
setScreenAnswer(pAnswer)	Splits pAnswer on ":" and applies the first to parts to the hour and minute input elements, respectively.
validate(pAnswer)	Validates the hours and minutes of pAnswer to be numerical and within their proper ranges.
Other methods	
-	-

ITPSelectQuestion

A descendant of ITPQuestion that represents any select question on a form. Base class for more specific select question representations. Transforms any select element using the ITPSelect widget. Uses the functions IsTextBlockSelect and allowsOrdering to parametrize the widget behavior.

Constructor	
ITPSelectQuestion	Trivial
Public methods	
-	-
Overrides	

initScreenElements()	Creates an itpselect widget out of the select element produced by the Xslt. Unless the control is a simple single select, which should result in a simple drop-down box. Sets the attributes textblocks and alloworder, based on the results of the isTextBlockSelect() and allowsOrdering() methods.
setInternalAnswer(pAnswer)	Synchronizes with the hidden input element that represents the internal value.
getScreenAnswer()	Returns the stored screenvalue.
setScreenAnswer(pAnswer)	Sets the stored screenvalue. **TODO**: figure out how to communicate with the widget about this.
synchronize()	If the page is a true Content Wizard form, sends a message and processes the response in order to rearrange the Content Wizard page.
Other methods	
isTextblockSelect()	Simply returns false. To be overridden by descendant classes.
allowsOrdering()	Simply returns false. To be overridden by descendant classes.

ITPSingleSelectQuestion

Descendant of ITPSelectQuestion that represents a single select question. Base class for all single select questions. Does not add any functionality.

Constructor	
ITPSingleSelectQuestion	Trivial
Public methods	
-	-
Overrides	
-	-
Other methods	
-	-

ITPSimpleSingleSelectQuestion

Descendant of ITPSingleSelectQuestion that represents a simple single select question, such as no radio buttons and no text blocks.

Constructor	
ITPSimpleSingleSelectQuestion	Trivial

Public methods	
-	-
Overrides	
-	-
Other methods	
-	-

ITPRadioSingleSelectQuestion

Descendant of ITPSingleSelectQuestion that represents a radio single select question.

Constructor	
ITPRadioSingleSelectQuestion	Trivial.
Public methods	
-	-
Overrides	
-	-
Other methods	
-	-

ITPTextblockSingleSelectQuestion

Descendant of ITPSingleSelectQuestion that represents a text block single select question.

Constructor	
ITPTextblockSingleSelectQuestion	Trivial.
Public methods	
-	-
Overrides	
-	-
Other methods	
isTextblockSelect()	Returns true.

ITPMultiSelectQuestion

Descendant of ITPSelectQuestion that represents a multi select question. Base class for all multi select questions.

Constructor	
ITPMultiSelectQuestion	Trivial
Public methods	
selectAnswer(pAnswer, pSelect)	Select or unselect one specific answer programmatically
Overrides	
initScreenElements	Add hint that items can be ordered through dragging.
Other methods	
allowsOrdering()	Returns true if and only if the info structure holds an orderresponse attribute.

ITPSimpleMultiSelectQuestion

Descendant of ITPMultiSelectQuestion that represents a simple multi select question, such as no radio buttons and no text blocks.

Constructor	
ITPSimpleMultiSelectQuestion	Trivial
Public methods	
-	-
Overrides	
-	-
Other methods	
-	-

ITPTextblockMultiSelectQuestion

Descendant of ITPMultiSelectQuestion that represents a text block multi select question.

Constructor	
ITPTextblockMultiSelectQuestion	Trivial.
Public methods	
-	-

Overrides	
-	-
Other methods	
isTextblockSelect()	Returns true.