APIReference

1. iKnowBase API Reference

Welcome to iKnowBase! This is the on-line documentation for iKnowBase. Please enjoy!

1. iKnowBase API Reference 2
2. API Reference 6
3. FreeMarker Templates 7
    1. About FreeMarker 7
    2. Includes and imports 7
    3. FreeMarker Directives 7
    4. Built-in Functions 10
        1. Built-ins for strings 10
        2. Built-ins for numbers 10
        3. Built-ins for dates 10
4. Groovy Templates 12
    1. About Groovy 12
    2. Where can you use Groovy 12
    3. Implicit objects 12
    4. Examples 13
5. FreeMarker model reference 15
    1. Understanding the notation 15
    2. Wrapped objects 15
6. Generic models 16
    1. ObjectReferenceModel 16
    2. ComponentModel 16
    3. ContextModel 16
    4. UserModel 17
    5. UserTokenModel 17
    6. DomainMode 17
    7. RequestModel 18
    8. ParametersModel 18
    9. ParameterModel 18
    10. TextMapModel 18
        1. TextMapModel 18
        2. TextStringModel 19
    11. LoadXML macro 19
    12. LoadURL macro 20
    13. AssignDate macro 20
7. Content viewer 21
    1. Overview 21
    2. Global objects 22
    3. ViewerModel 22
    4. RowSetModel 23
    5. RowModel 24
    6. DocumentModel 25
    7. DataFields 25
    8. DataFieldString 26
    9. DataFieldNumber 26
    10. DataFieldDate 27
    11. DataFieldClob 27
    12. DataFieldFiles 28
    13. DataFieldFile 28
    14. DataFieldReferences 29
    15. DataFieldReference 29
    16. DataFieldActions 29
    17. DataFieldAction 30
    18. AttributeModel 30
    19. StyleModel 30
    20. QuickLinkModel 31
    21. StyleFields 31

22. StyleField 31
23. Logic 32
24. Navigation 32
25. GotoModel 33
26. HtmlLink 34
27. NavigationLink 34
8. Content search 36
1. Global objects 36
2. ViewerModel 36
3. SearchModel 36
9. XML viewer 37
1. Overview 37
2. Global objects 37
3. ViewerModel 37
10. Dimension viewer 38
1. Overview 38
2. Global objects 38
3. ViewerModel 38
4. DimensionListModel 38
5. DimensionModel 38
6. DimensionActionModel 39
11. Menu viewer 40
1. Overview 40
2. Global objects 40
3. ViewerModel 40
4. MenuItemListModel 40
5. MenuItemModel 40
6. MenuItemActionModel 41
12. HTML Viewer 42
1. Overview 42
2. Global objects 42
13. BPEL Task form 43
1. Overview 43
2. Global objects 43
3. TaskFormModel 43
4. XMLFormModel 44
5. AttributesModel 44
6. AttributeModel 44
7. SubmitModel 44
8. Definition 45
14. BPEL Task viewer 46
1. Overview 46
2. Global objects 46
3. TaskModel 46
4. TaskDataModel 46
5. TaskFieldModel 47
6. TaskStyle 47
7. TaskStyleField 47
15. BPEL Process viewer 49
1. Overview 49
2. Global objects 49
3. ProcessModel 49
16. Activiti Form viewer 50
1. Overview 50
2. FormModel 50
3. FormDataModel 51
4. FormComponentModel 51
5. FormDirective 52
6. FormInputDirective 52
7. FormLabelDirective 53
8. FormButtonDirective 53

9. FormMessageDirective ... 54
17. Page ... 55
1. Overview ... 55
2. Global objects ... 55
3. PageModel ... 55
4. ResourcesModel ... 56
5. RegionsModel ... 56
6. RegionModel ... 57
7. PageComponentModel ... 58
18. Javascript library ... 59
1. Global objects ... 59
2. iKnowBase root object ... 59
1. busyIndicator ... 59
1. Parameters ... 59
2. Methods ... 60
3. Example ... 60
2. notify ... 60
1. Parameters ... 60
2. Methods ... 60
3. Example ... 61
3. iKnowBase.PageEngine ... 61
1. reloadComponent ... 61
1. Syntax and parameters ... 61
2. Finding the id of the page component ... 62
3. Deprecated syntax and parameters ... 63
4. iKnowBase.PageEngine.InstantContentCache ... 63
1. start ... 63
1. Syntax and parameters ... 63
5. iKnowBase.Instant ... 64
1. Atmosphere.subscribe ... 64
1. Syntax ... 64
2. Options ... 65
3. Topic name and topic options ... 66
4. Examples ... 67
2. Atmosphere.publish ... 68
1. Syntax ... 68
2. Example ... 68
3. Atmosphere.unsubscribe ... 68
1. Syntax ... 68
4. Atmosphere.requestUserListUpdate ... 69
1. Syntax ... 69
2. Requirements ... 69
3. Example ... 69
5. Atmosphere.requestUserList ... 70
1. Syntax ... 70
2. Requirements ... 70
3. Example ... 70
6. Atmosphere.requestSubscribeUserListChanges ... 71
1. Syntax ... 71
2. Requirements ... 71
3. Example ... 71
7. setDefaultOptions ... 72
1. Syntax ... 72
8. Complete example ... 72
19. Content Server ... 73
1. Download content ... 73
2. Download multiple files ... 74
3. Upload content ... 74
1. Existing documents ... 74
2. New documents ... 75
20. Page Engine ... 76

1. General use ... 76
2. Language support ... 76
3. Database trace functionality ... 76
4. Refresh content cache ... 76
21. Form HTML Template support ... 78
1. Overview of iKnowBase-specific tags ... 78
2. ikb:form ... 79
3. ikb:formname ... 79
4. ikb:label ... 79
5. ikb:input ... 79
6. ikb:dimension ... 79
7. ikb:favorite_select ... 79
8. ikb:space ... 80
9. ikb:calendar ... 80
10 ikb:radio ... 80
11 ikb:check ... 80
12 ikb:select ... 80
13 ikb:button ... 80
14 ikb:related_object ... 81
15 ikb:related_image ... 81
16 ikb:file ... 82
17 ikb:filelink ... 82
18 ikb:fileurl ... 82
19 ikb:attachment ... 82
20 ikb:textarea ... 82
21 ikb:imagevariant ... 82
22 ikb:addattribute ... 82
23 ikb:display_add_attr ... 83
24 ikb:prompt ... 83
25 ikb:display ... 83
26 ikb:conditional ... 83
27 ikb:masterselect ... 83
28 ikb:condition ... 83
29 oracle ... 83
22. SOLR Schema model ... 85
1. Overview of iKnowBase fields in solr/Schema.xml ... 85
23. Instant Server ... 87
1. HTTP API ... 87
1. /ikb$service/v1/sendMessage: ... 87
2. /private/ikb$service/v1/sendMessage: ... 88
3. Sending messageFormat=IKB ... 88
4. /ikb$service/v1/requestUserList: ... 88
5. /private/ikb$service/v1/requestUserList: ... 88
6. /ikb$service/v1/requestUserListUpdate: ... 88
7. /private/ikb$service/v1/requestUserListUpdate: ... 89
2. PLSQL API ... 89
1. Publish message ... 89
2. Server request: requestUserList ... 90
3. Server request: requestUserListUpdate ... 91

2. API Reference

This part of the book contains API references for various APIs you may want to use while developing an application using iKnowBase.

3. FreeMarker Templates

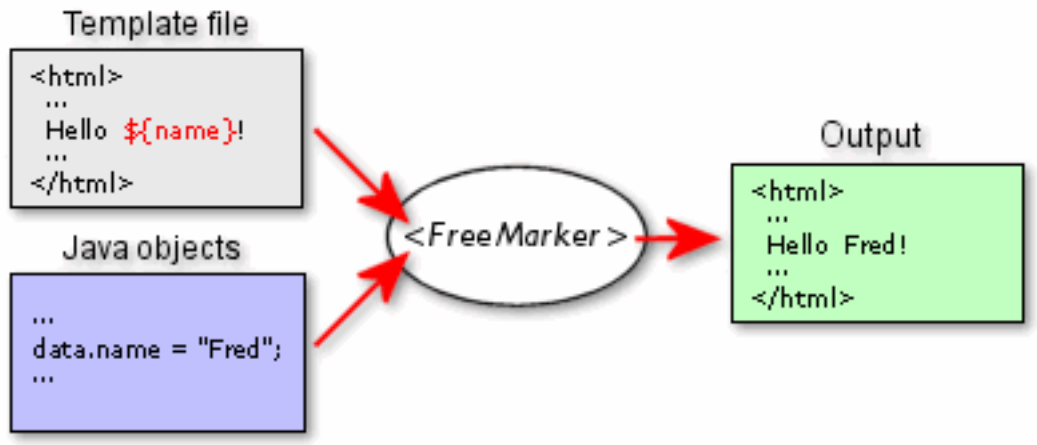
iKnowBase supports the use of FreeMarker templates. When using FreeMarker templates, you will combine static markup, standard FreeMarker functionality and iKnowBase model objects to create the desired output:

- Static markup is the static text that you want output from the template. This is typically HTML, but can really be any kind of text that you want, for example to generate XML, RSS or JSON.
- iKnowBase model objects are the data that iKnowBase makes available to you.
- Standard FreeMarker functionality are directives and expressions that allow you to iterate over, print and otherwise manipulate the model objects.

In order to best utilize the FreeMarker technology, you should read the FreeMarker documentation available at http://freemarker.org/docs. Some key features are described below, but this covers only a small subset of the capabilities.

About FreeMarker

With FreeMarker you can create HTML-based templates that can display data from a Java model. The following figure shows how FreeMarker connects with a template (in this instance an HTML template), and data from a Java object model. This connection generates an HTML output that displays the data in the template.



Includes and imports

FreeMarker supports `<#include>` and `<#import>` as mechanisms to reuse templates. With iKnowBase, it is possible to use both of these mechanisms to refer to templates stored inside the iKnowBase repository:

- First, create an iKnowBase template in iKnowBase Development Studio. Take note of the GUID of the template, listed at the bottom of the screen.
- Next, refer to this template using the special filename “/urn:iknowbase:template:guid:” + guid, for example `<#include "/urn:iknowbase:template:guid:9E23B9E919E73ECCE040000A18006DD1">`

FreeMarker Directives

FTL tags are used to call directives. There are two types FTL tags:

- Start-tag: `<#directive name parameters>`
- End-tag: `</#directive name>`

These tags are similar to the syntaxes in the HTML code and the XML code except for tags that start with #. There are two types of directives: predefined directives and user-defined directives. User defined directives use @ instead of #. For example, <@mydirective parameter>... </@mydirective>.

The format of the parameters depends on the name of the directive.

Property	Description
if, else, elseif	Syntax: <#if condition> ... <#elseif condition2> ... <#elseif condition3> ... <#else> ... </#if> For example: <#if x == 1> x is 1 <#elseif x == 2> x is 2 <#else> x is neither 1 nor 2 </#if> ikbProperty. switch, case, default, break Syntax: <#switch value> <#case refValue1> ... <#break> <#case refValue2> ... <#break> ... <#case refValueN> ... <#break> <#default> ... </#switch> For example: <#switch x> <#case x = 1> x is 1 <#case x = 2> x is 2 <#default> x is neither 1 nor 2 </#switch>

```
ikbProperty. list, break

This directive is used to loop through
 a list of items:


    <#list sequence as item>
        Item = ${item}
    </#list>

p.For example:


    <#assign seq = ["winter",
 "spring", "summer", "autumn"]>
    <#list seq as x>
        ${x_index + 1}. ${x}<#if
 x_has_next>,</#if>
    </#list>

ikbProperty. Assign

This directive is used to create a
 variable:


<#assign name=value>

or

    <#assign name>
    This is the value assigned to name
    </#assign>


For example:

<#assign seasons = ["winter",
 "spring", "summer", "autumn"]>
```

If you want to write values directly within the insertion brackets ${ and }, there are some characters that need a specific syntax.

Syntax	Description
\"	Quotation mark (u0022)
\'	Apostrophe (apostrophe-quote) (u0027)
\\	Back slash (u005C)
\n	Line feed (u000A)
\r	Carriage return (u000D)
\t	Horizontal tabulation (tab) (u0009)
\b	Backspace (u0008)
\f	Form feed (u000C)
\l	Less than sign: <
\g	Greater than sign: >
\a	Ampersand: &
\xCode	Character given with its hexadecimal Unicode (UCS) code

Built-in Functions

FreeMarker contains a set of built-in functions that you can use directly with variables.

Built-ins for strings

Function	Syntax and Description
html	The ${string?html} variable is coded to the HTML text. For example, the character < becomes &lt.
js_string	The ${string?js_string} variable is coded to the JavaScript text. For example, the character ‘ becomes ''.
xml	The ${string?xml} variable is coded to the XML text. For example, the character > becomes &gt.
url	The ${string?url} variable is coded to the URL text. For example, the character / becomes %2b.
cap_first	The ${string?cap_first} variable has the initial character capitalized.
lower_case	The ${string?lower_case} variable gets all the characters decapitalized.
upper_case	The ${string?upper_case} variable gets all the characters capitalized.
trim	The ${string?trim} variable peels off all blank characters at the start and end.
substring	This function uses ${string?substring(from, [to])} tag to take a segment of the variable depending on specified positions.
replace	The ${string?replace(“this”, “with that”)} variable substitutes the text with other text.
number	This function uses ${string?number} tag to convert a variable from string to number.

Built-ins for numbers

Function	Syntax and Description
string	This function uses ${number?string} tag to convert a variable from number to text.
string.number	This function uses ${number?string.number} tag to convert a variable from number to text.
string.percent	This function uses ${number?string.percent} tag to convert a variable from number to percent text.

Built-ins for dates

Function	Syntax and Description
string	This function uses ${date?string} tag to convert a variable from date to text.
string.short	This function uses ${date?string.short} tag to convert a variable from date to a predefined short text.
string.medium	This function uses ${date?string.medium} tag to convert a variable from date to a predefined medium text.

string.long	This function uses ${date?string.long} tag to convert a variable from date to a predefined long text.
string.full	This function uses ${date?string.full} tag to convert a variable from date to a predefined full text.

4. Groovy Templates

iKnowBase supports the use of Groovy templates. When using Groovy templates, you will use the power of groovy to create the desired output, or perform the required actions.

combine static markup, standard FreeMarker functionality and iKnowBase model objects to create the desired output:

- Static markup is the static text that you want output from the template. This is typically HTML, but can really be any kind of text that you want, for example to generate XML, RSS or JSON.
- iKnowBase model objects are the data that iKnowBase makes available to you.
- Standard FreeMarker functionality are directives and expressions that allow you to iterate over, print and otherwise manipulate the model objects.

In order to best utilize the FreeMarker technology, you should read the FreeMarker documentation available at http://freemarker.org/docs. Some key features are described below, but this covers only a small subset of the capabilities.

About Groovy

Groovy is a full-featured programming language for the Java Virtual Machine. More information about groovy is available at the following sources:

- http://groovy.codehaus.org/
- http://groovy.codehaus.org/Documentation
- JavaDoc for Groovy internal classes written in Java
- GroovyDoc for Groovy internal classes written in Groovy

Where can you use Groovy

Groovy is available in many places in iKnowBase.

- Anywhere there is a presentation template, you can choose between Groovy and FreeMarker. They have access to the same underlying data, but with different syntaxes and capabilities.
- As an action (similar to servlets), connected directly to a Target (URL).

Implicit objects

When using Groovy, iKnowBase will always provide you with a number of “implicit objects” available to the script. These are documented in the API Reference, under each component type, and are the same objects that are being made available to FreeMarker templates. However, Groovy templates have access to a number of additional objects.

For templates that run in any sort of viewer or page, the following objects are available:

Object	Description
out	A PrintWriter representing the output destination
html	A MarkupBuilder useful for creating HTML or XML markup to the output destination
json	A StreamingJsonBuilder useful for creating JSON markup to the output destination
iknowbase	A RepositoryModel providing access to the iKnowBase database repository

For templates that run as a Script Target, the following additional objects are also available:

Object	Description
sout	A ServletOutputStream representing the output destination

Examples

This chapter contains a couple of very small examples, indicating how you can use Groovy along with iKnowBase.

This example shows a couple of items. First, how you can use the iknowbase-object to get a Groovy SQL object, and then use that to query the database. Second, it shows how the implicit MarkupBuilder named *html* can be used to produce well-formatted html output.

```
 // Prepare some data
def pageTitle = "/SYSTEST/Action/GroovyScript-HtmlBuilder"
def users = iknowbase.sql.rows("select * from all_users where rownum <= 5")

// Output a complete html-document
html.html {
   head {
      title(pageTitle)
      style(type:"text/css","""
      table { border:thin solid black; width:100%; }
      th { background-color: #999 }
      """)
   }
   body {
      h1(pageTitle)
      table {
         tr {
            th("User ID")
            th("Username")
         }
         users.each { user->
            tr {
               td(user.user_id)
               td(user.username.toLowerCase())
            }
         }
      }
   }
}
```

This example shows how the implict StreamingJsonBuilder named *json* can automate the production of json:

```
def dbUsers = iknowbase.sql.rows("select * from all_users where rownum <= 5")

// Output a complete json-document
def userlist = [users:dbUsers.collect { [id:it.user_id,
 username:it.username.toLowerCase() ] }]
def userinfo = [userinfo:userlist]

json(userinfo)
```

This example shows how to use the documentService to create a document from an activitiy Script task.

```
import com.iknowbase.api.contentservices.v2.model.*;
def userref = new UserReference().withUsername(initiator)

def document = new Document()
document.documentIdentity = new DocumentReference(null, null, "activiti_" +
 execution.getProcessInstanceId(), null, null)
document.title = "Activiti Process - ProcessInstanceId: " +
 execution.getProcessInstanceId()
```

```
document.documentTypeReference = new ObjectReference(null, null, "TEMPDATA",
 null);

// Create ikb document
def documentReference = iknowbase.documentService.saveDocument (
   userref,
   document,
   SaveOperationEnumeration.INSERT,
   SaveOperationEnumeration.NONE,
   SaveOperationEnumeration.NONE,
   null, null, null, false
);
```

5. FreeMarker model reference

This chapter contains a reference for all the iKnowBase-specific FreeMarker model objects, with a description of their properties.

Every component which supports FreeMarker templates has its own section in this chapter, with a separate chapter for shared model object at the end.

Understanding the notation

For every component, there are two pieces of information:

- The first table inside the chapter describes the objects made available to the FreeMarker template. For each object there is a short description, as well as information about the type of the model object.
- Subsequent tables describe the properties of each model object type. For each property there is a short description, as well as information about the type of the property.

Often, there is a reference to a property named `<default>`. This is the value returned from the model object when no property is specified.

Also, there are often references to a property named `<name>`. These refer to a “generic” notation, where you can use any name. This name is then typically used to look up information for an object of that name.

Wrapped objects

Some of the models may refer to *wrapped object* properties, with reference to a java class. In those situations, the properties of that model object will be defined by the java class directly, through the use of FreeMarker’s BeanModel. For detailed information, see the FreeMarker documentation of the BeansModel class.

6. Generic models

These models are used across components and portlets.

ObjectReferenceModel

This object contains a single reference to an external object (value list, dimension, etc).

Property	Description	Type
label	The label value of the reference object	String
objectId	The object id of the references object	Number
objectGuid	The object guid of the referenced object	String
externalKey	The external_key specified for the attribute behind the field	String
`<default>`	When used directly, this object evaluates to the “label” property.	String

ComponentModel

This object contains information about the currently execution component (the ContentViewer).

Property	Description	Type
id	Name of the component as it is known on the client side (in the web browser). Use this for ajax-based api calls on the component.	String

Example:

```
function refreshThisComponent() {
   iKnowBase.PageEngine.reloadComponent ('${component.id}');
}
```

ContextModel

The ContextModel-object contains general information related to an execution context. Use this object to find information about current time and current user.

Example:

```
<p>
This information was last refreshed on ${context.datetime}
</p>
```

Property	Description	Type
user	Information about the executing user, either a logged on user or the guest user applicable to the request	UserModel
domain	Information about the domain applicable to the request	DomainModel
date	Current date	Date
time	Current time	Time

datetime	Current date and time	Datetime
millis	Number of milliseconds since 1.1.1970	Number
request	Information about the current request	RequestModel
language	iKnowBase language code for the current context.	String
isDevelopmentMode	Flag indicating whether development mode is enabled or not	Boolean

UserModel

The UserModel-object contains information about the executing user, either a logged on user or the guest user applicable to the request.

Property	Description	Type
isLoggedOn	Whether the current user is logged on, or is a public user	Number
id	Numeric user id for the user	Number
username	Username (loginname) for the user	String
token	UserToken for secure authentication without logging in, for ajax-based processing.	UserTokenModel

UserTokenModel

The UserTokenModel-object contains a token for secure authentication without logging in, for ajax-based processing.

Property	Description	Type
name	Name of the URL-parameter for the token	String
value	Token value	String

DomainMode

The DomainModel-object contains information about the domain applicable to the current request.

Property	Description	Type
objectId	The object ID of the domain	Number
objectGuid	The object guid of the domain	String
label	The label of the domain	String
name	The name of the domain, typically HOST.PORT	String
ikbViewerPath	The path to the ikbViewer application, without a trailing slash	String
pageEnginePath	The path to the page engine, without a trailing slash/ikbViewer	String
contentServerPath	The path to the content server, without a trailing slash	String
resourcePath	The path to the resource directory, without a trailing slash	String

ikbStudioPath	The path to the development studio, without a trailing slash	String

RequestModel

This object contains parameters available to the component.

Property	Description	Type
pageurl	URL for the page currently running.	String
servername	Name of server where the viewer is running (part of pageurl)	String
serverport	Port number of server where the viewer is running (part of pageurl)	String
parameters	Parameters sent to the request.	ParametersModel
`<name>`	Value of URL-parameter (shortcut to `parameter.name`).	String

The properties username, referencepath and guid were earlier available in this model. They have been deprecated.

ParametersModel

This object contains information about request parameters from the current HTTP-request. The model supports several of the basic FreeMarker model types:

- It is a TemplateHashModel, supporting one property per parameter
- It is a TemplateHashModelEx, supporting the built-ins `?keys` and `?values`.

Property	Description	Type
`<name>`	Value of URL-parameter (shortcut to `parameter.name`).	ParameterModel, String

ParameterModel

This model contains all values for a given parameter. The model supports several of the basic FreeMarker model types:

- It is a String (ScalarModel), returning the first (or only) value for the parameter
- It is a TemplateSequenceModel containing Strings, supporting index-based lookups (`request.parameters.paramname[index]`) and iteration (`...[/#list]`).

TextMapModel

This model object contains language specific texts defined with various iKnowBase objects. The model object is a compound object. First is the actual TextMapModel, which enables you to find a TextString based on a user defined key. Next is the TextStringModel, which decides which definition of the string to use, for example the one stored with a Page or the one stored with a Template.

You may use only the TextMapModel object (as in @${viewer.strings.contactEmail}), which will then search for the string in any of the available locations. You may also specify the location “any” for the same purpose (as in @${viewer.strings.contactEmail.any}). If you want complete control of the string location, specify it directly (as in @${viewer.strings.contactEmail.template}).

The locations available vary from portlet to portlet. See the definition of the portlet object for description of which locations apply to that portlet.

TextMapModel

Property	Description	Type

<name>	Text string definition.	TextStringModel

TextStringModel

Property	Description	Type
page	The value of the text with the given name, as specified on the page.	String
template	The value of the text with the given name, as specified on the template.	String
style	The value of the text with the given name, as specified on the presentation style.	String
viewer	The value of the text with the given name, as specified on the viewer.	String
domain	The value of the text with the given name, as specified on the domain.	String
any	Return any string, from any of the available locations (properties).	String
<default>	Value from one of style, viewer and domain, if present; otherwise null.	String

Examples:

```
<!-- The text specified in the "contactEmail" string, in any available
 location. -->
${viewer.strings.contactEmail}

<!-- The text specified in the "contactEmail" string, in any available
 location. -->
${viewer.strings.contactEmail.any}

<!-- The text specified in the "contactEmail" string, on the template only --
>
${viewer.strings.contactEmail.template}

<!-- The freemarker default value operator does not work without a location
 -->
${viewer.strings.contactEmail!"default-email@example.com"}

<!-- The freemarker default value operator works only when specifying a
 location -->
${viewer.strings.contactEmail.any!"default-email@example.com"}
${viewer.strings.contactEmail.template!"default-email@example.com"}
```

LoadXML macro

Use this macro to load XML from a specified URL, for further processing using FreeMarker.

Parameter	Description	Type
name	Name of variable you want to create, where the XML will be loaded into	String
url	URL where the specified XML resides	String

timeout	Optional timeout in milliseconds. If no timeout is specified, a default value of 60 seconds will be used.. The timeout applies to each of the connect and read phases independently.	Number

Examples:

```
<@ikb.loadXML 'myxml' 'http://www.example.com/xmlfeed' />
<#list ikb.myxml.entries as entry>
    ${entry.title}
</#list>

<@ikb.loadXML 'myxml' 'http://www.example.com/xmlfeed' 2000 />
<#if ikb.myxml! == "">
   No data was returned; possibly timeout
</#if>
```

LoadURL macro

Use this macro to load arbitrary text from a specified URL, for further processing using FreeMarker.

Parameter	Description	Type
name	Name of variable you want to create, where the content will be loaded into	String
url	URL where the specified content resides	String
timeout	Optional timeout in milliseconds. The timeout applies to each of the connect and read phases independently	Number

Example:

```
<@ikb.loadURL 'myurl' 'http://my.domain.com/htmlfeed '/>
<div>
    ${ikb.myurl}
</div>
```

AssignDate macro

Use this macro for simple date arithmetic, where you start with a certain date and then add a duration.

Parameter	Description	Type
name	Name of variable you want to create, where the content will be loaded into	String
base	Base date	Datetime
increment	Increment in milliseconds	Number

Example:

```
<@ikb.assignDate 'dueWarning' viewer.logic.currentDate 259200000 />
<div>
    ${ikb.dueWarning}
</div>
```

7. Content viewer

Overview

The figure below gives a quick overview of the models available to the ContentViewer.

- viewer
  - data
    - isEmpty
    - count
    - lastLevel
    - level/recurse
      - fields
        - <name> (String, Number, Date)
          - text
          - date
          - number
          - externalKey
          - hasValue
          - label
          - style
          - xml
          - url
          - script
          - nodeText
      - documentLevel
      - documentId
      - isFirst
      - isLast
      - isOdd
      - isEven
      - sub
        - data
        - style
- style
  - fields
    - <name>
      - label
      - isSortActive
      - isSortActiveAscending
      - isSortActiveDescending
      - sortAction
      - sortActionAscending
      - sortActionDescending
      - sortIconActive
      - sortIconAscending
      - sortIconDescending
  - hasQuickLink
  - quickLink
    - text
    - url
    - script
    - nodeText
  - portletTitle
- logic
  - languageId
  - noRecordsFound
  - currentDate
- navigation
  - pageSize

    - firstRow
    - totalRows
    - startRow
    - maxRow
    - lastRow
    - text
    - isEmpty
    - range
    - first
    - previous
    - next
    - last
    - goto
    - showMore
- param
    - username
    - pageurl
    - referencepath
    - servername
    - serverport
    - guid
    - `<name>`
- strings
    - `<name>`
        - style
        - viewer
        - domain
- context
    - user
        - id
        - username
        - token
            - name
            - value
    - date
    - time
    - datetime
    - millis
- request
    - pageurl
    - servername
    - serverport
- component
    - id

Global objects

These are the global (top level) objects available when using FreeMarker from a ContentViewer.

Object	Description	Type
viewer	This object contains information about the viewer, its presentation style and its data.	ViewerModel
context	This object contains information about the execution context.	ContextModel
component	This object contains information about the executing component.	ComponentModel

ViewerModel

This object contains information about the ContentViewer itself, its definition and data.

Property	Description	Type
data	The full data set of the content viewer.	RowSetModel
style	The presentation style of the content viewer.	StyleSet
logic	Logic information about the content viewer.	LogicModel
navigation	Navigational information about the content viewer.	NavigationModel
param	URL-parameters.	ParameterModel
strings	Available strings for the viewer. This TextMapModel supports locations “style”, “viewer” and “domain”.	TextMapModel
`<default>`	When used directly, this object evaluates to the value of the “data” property.	RowSetModel

RowSetModel

This object contains a set of rows/documents. The top level RowSetModel (as returned from `viewer.data`) contains all rows/documents returned from the content viewer, whereas RowSetModels accessed differently (for example through `row.sub` from a specific row) contains only a partial set of documents.

The object is a freemarker sequence model, so you can access the nodes directly using `${data[index]}`, or you can traverse it using the `...[/#list]` syntax.

The object is also a freemarker node, so you can also use the built-ins ?children, ?parent, ?root, ?ancestors, ?node_name, ?node_type and ?node_namespace. The ?node_name and ?node_type properties are both “DocumentList”.

Property	Description	Type
isEmpty	A flag indicating whether the content viewer returned any rows. Returns true if the viewer is empty.	Boolean
count	The number of documents on the current level (documentLevel)	Number
lastLevel	The level of the last document available in this data set.	Number
level	A sequence containing only documents on this particular level, e.g. without subdocuments. This is the default, so it can be skipped.	Sequence(Row)
recurse	A sequence containing both documents on this level **and** all subdocuments.	Sequence(Row)
style	The presentation style of the current level of the ContentViewer.	StyleSet
?children	List of documents	List of RowModel
?parent	There is no parent; it always returns null.	Null

?root	There is no root node; it always returns null.	RowModel
?ancestors	List of ancestors	List of RowModel
?node_namespace	There is no namespace; it always returns null.	Null
?node_type	Always the string “DocumentList”	String
?node_name	Always the string “DocumentList”	String
`<default>`	When used directly, the rowset object evaluates to a sequence of Row-objects.	Sequence(Row)
`<name>`	The data field object for a given field. The field name refers to “reference name” from the presentation style definition. The type of the property varies, depending on the data type of the data field.	DataFieldXxx

RowModel

This model object contains information about a single row / document returned from the viewer.

The model is a freemarker node, so you can also use the built-ins ?children, ?parent, ?root, ?ancestors, ?node_name, ?node_type and ?node_namespace.

Note that part of the information is context sensitive. For example, a document might be the second document in the collection `data.viewer.level` (and thus have isOdd=false), while also being available as the third document in the collection `data.viewer.recurse` (and thus have isOdd=true).

Property	Description	Type
fields	The fields (attributes) from a single row/document in the data set of the ContentViewer.	DataFields
documentLevel	The level of the current document.	Number
documentId	The documentId of the current document.	Number
isOdd	The current document is on an odd row number (1,3,5,...)	Boolean
isFirst	The current document is the first in the result set.	Boolean
isLast	The current document is the last in the result set.	Boolean
sub	A collection of sub documents from the current document.	RowSetModel
document	The returned document, without attributes	DocumentModel
?children	List of children documents	List of RowModel
?parent	Parent node	RowModel
?root	Root node	RowModel
?ancestors	List of ancestors	List of RowModel
?node_namespace	There is no namespace; it always returns null.	Null
?node_type	Always the string “Document”	String

?node_name	Always the string “Document”	String
<default>	When used directly, this object evaluates to the “fields” property.	
<name>	The data field object for a given field. The field name refers to “reference name” from the presentation style definition. The type of the property varies, depending on the data type of the data field.	DataFieldXxx

DocumentModel

This object represents an iKnowBase document. The structure of the DocumentModel is more or less the same as the “Document” object in the ServiceAPI.

Note that this model may not be fully populated. In particular, the various ObjectReferences will probably only contain an id or a guid, and the list of attributes is often not present.

Property	Description	Type
documentReference	Document identity, with id, guid and externalKey	ObjectReferenceModel
title	Document title (same as documentReference.label)	ObjectReferenceModel
description	Document description (ingress)	String
aclReference	Reference to acl	ObjectReferenceModel
documentTypeReference	Reference to document type	ObjectReferenceModel
ownerReference	Reference to document owner	UserReferenceModel
parentReference	Reference to parent document	ObjectReferenceModel
statusReference	Reference to document status	ObjectReferenceModel
validFrom	“Valid from” attribute of document	Date
validTo	“Valid to” attribute of document	Date
created	Created-time of document	Date
updated	Updated-time of document	Date
isFile	Indicates whether the document content is a file	Boolean
isUrl	Indicates whether the document content is an URL	Boolean
isText	Indicates whether the document content is text (plain or html)	Boolean
isXml	Indicates whether the document content is XML	Boolean

DataFields

This object is a FreeMarker “hash” container containing the data fields for a given document in the ContentViewer. The object can be accessed by field name, as defined in the presentation style defintion.

Property	Description	Type
<name>	The data field object for a given field. The field name refers to “reference name” from the presentation style definition. The type of the property varies,	DataFieldXxx.

	depending on the data type of the data field.	

DataFieldString

This object contains the data field of type String.

Property	Description	Type
char	The unformatted string value of the field.	String
html	The formatted html value of the field.	String
text	The formatted html value of the field (deprecated for ".html")	String
label	The label value of the field	String
attribute	The attribute behind the field, or null if there is no attribute	AttributeModel
externalKey	(Deprecated, use attribute.externalKey)	String
hasValue	A boolean indicating whether the value is specified and not blank.	Boolean
style	The style specification for this particular data field.	Style
xml	For non-system attributes, return the string value as a parsed XML Node object.	XML
url	For system attributes that represent a link, this returns the value of the "href" attribute of the <a>-tag.	String
onclick	For system attributes that represent a link, this returns a javascript-statement that can be used to follow the link. The onclick event must be attached to an <a>-tag, since the returned statement refers to the containing <a>-tag.	String
script	This property is deprecated; use the onclick-property instead.	String
nodeText	(Deprecated) For system attributes that return a link, this returns the text elements inside the <a>-tag.	String
nodeValue	(Deprecated) For system attributes that return a link, this returns the text elements inside the <a>-tag.	String
<default>	When used directly, this object evaluates to the "text" property.	String

DataFieldNumber

This object contains the data field of type Number.

Property	Description	Type

number	The number value of the field.	Number
char	The unformatted string value of the field.	String
html	The formatted html value of the field.	String
text	The formatted html value of the field (deprecated for ".html")	String
label	The label value of the field	String
attribute	The attribute behind the field, or null if there is no attribute	AttributeModel
externalKey	(Deprecated, use attribute.externalKey)	String
hasValue	A boolean indicating whether the value is specified and not blank.	Boolean
style	The style specification for this particular data field.	Style
`<default>`	When used directly, this object evaluates to the "number" property.	Number

DataFieldDate

This object contains the data field of type Number.

Property	Description	Type
date	The date value of the field.	Date
char	The unformatted string value of the field.	String
html	The formatted html value of the field.	String
text	The formatted html value of the field (deprecated for ".html")	String
label	The label value of the field	String
attribute	The attribute behind the field, or null if there is no attribute	AttributeModel
externalKey	The external_key specified for the attribute behind the field	String
hasValue	A boolean indicating whether the value is specified and not blank.	Boolean
style	The style specification for this particular data field.	Style
`<default>`	When used directly, this object evaluates to the "date" property.	Date

DataFieldClob

This object contains the data field of type Clob.

Property	Description	Type
char	The unformatted string value of the field.	String
html	The formatted html value of the field.	String

text	The formatted html value of the field (deprecated for ".html")	String
label	The label value of the field	String
attribute	The attribute behind the field, or null if there is no attribute	AttributeModel
externalKey	The external_key specified for the attribute behind the field	String
hasValue	A boolean indicating whether the value is specified and not blank.	Boolean
style	The style specification for this particular data field.	Style
`<default>`	When used directly, this object evaluates to the "text" property.	String

DataFieldFiles

This object contains the data field of type Files, e.g. where the value is one or more files. This is of course used for attachments, but also for derived image formats (such as thumbnails) for image documents.

Property	Description	Type
char	The unformatted string value of the field.	String
html	The formatted html value of the field.	String
text	The formatted html value of the field (deprecated for ".html")	String
label	The label value of the field	String
attribute	The attribute behind the field, or null if there is no attribute	AttributeModel
externalKey	The external_key specified for the attribute behind the field	String
hasValue	A boolean indicating whether the value is specified and not blank.	Boolean
style	The style specification for this particular data field.	Style
...	Any other property maps to the same property on the first File, so that ${file.xxxx} maps to {file[0].xxx}	...
`[]`	This object is a sequence of DataFieldReference objects.	DataFieldReference
`<default>`	When used directly, this object evaluates to the "text" property.	String

DataFieldFile

This object represents a File value.

Property	Description	Type
url	The URL of the file content	String
mimetype	The mimetype of the file, if known	String
filename	The filename of the file, if known	String

DataFieldReferences

This object contains the data field of type References, e.g. where the value is one or more references to external objects (value lists, dimensions, etc).

Property	Description	Type
char	The unformatted string value of the field.	String
html	The formatted html value of the field.	String
text	The formatted html value of the field (deprecated for ".html")	String
label	The label value of the field	String
attribute	The attribute behind the field, or null if there is no attribute	AttributeModel
externalKey	The external_key specified for the attribute behind the field	String
hasValue	A boolean indicating whether the value is specified and not blank.	Boolean
style	The style specification for this particular data field.	Style
...	Any other property maps to the same property on the first Refernce	...
`[]`	This object is a sequence of DataFieldReference objects.	DataFieldReference
`<default>`	When used directly, this object evaluates to the "text" property.	String

DataFieldReference

This object contains an ObjectReference, e.g. a reference to an object in the iKnowBase repository. Exactly what is being pointed to cannot be found from this object; this must be derived from the context (such as the containing attribute): A value list attribute would contain a reference to a value list, while a dimension attribute would contain a reference to a dimension.

Property	Description	Type
label	The label value of the reference	String
objectId	The ID of the reference	Number
objectGuid	The guid of the reference	String
externalKey	The external key of the reference	String

DataFieldActions

This object is used for data fields that represent actions, such as edit and delete links. It is also sometimes used for data fields that link to other objects, such as document and image links.

Property	Description	Type
char	The unformatted string value of the field.	String
html	The formatted html value of the field.	String
text	The formatted html value of the field (deprecated for ".html")	String
label	The label value of the field	String

attribute	The attribute behind the field, or null if there is no attribute	AttributeModel
externalKey	The external_key specified for the attribute behind the field	String
hasValue	A boolean indicating whether the value is specified and not blank.	Boolean
style	The style specification for this particular data field.	Style
...	Any other property maps to the same property on the first Action	...
`[]`	This object is a sequence of DataFieldAction objects.	DataFieldAction
`<default>`	When used directly, this object evaluates to the “text” property.	String

DataFieldAction

This object represents a single action, such as an edit or delete link.

Property	Description	Type
label	The label value of the action	String
url	The URL of the action, used as the “href” attribute of the <a>-tag.	String
onclick	The value of the “onclick” attribute of the <a>-tag.	String
script	The value of the “onclick” attribute of the <a>-tag.	String

AttributeModel

This object contains information about a defined attribute. Most DataField model objects have a reference to an AttributeModel object.

Property	Description	Type
objectId	The ID of the attribute	Number
objectGuid	The guid of the attribute	String
level	The level of the attribute	Number
label	The text of the attribute	String
name	The name of the attribute	String
datatype	The data of the attribute, as defined in the iKnowBase ServiceAPI. (DATE, NUMBER, CHAR, CLOB, DIMENSION, VALUELIST, DOCUMENT_LINK, PICTURE_LINK, EXTERNAL, DOCUMENT_FIELD or FILE).	String

StyleModel

This object contains the presentation style definition for a ContentViewer, or for a specific level inside the Content Viewer.

Property	Description	Type
fields	The fields defined inside a presentation style	StyleFields

hasQuickLink	A boolean indicating whether there are quicklinks	Boolean
quickLink	A list of quicklinks	TemplateSequenceModel<QuickLinkModel>
portletTitle	The title of the portlet	String
`<default>`	When used directly, this object evaluates to the “fields” property.	
`<name>`	The StyleField object for a given field. The field name refers to the “reference name” from the presentation style definition.	StyleField

QuickLinkModel

This object represents a quick link, as defined in a ContentViewer.

Property	Description	Type
text	The full html value of the quicklink, e.g. the full <a>-tag including attributes and content.	String
label	The label value of the field	String
url	The URL to the quick link target	String
onclick	The value of the “onclick” attribute of the <a>-tag.	String
script	The value of the “onclick” attribute of the <a>-tag.	String
nodeText	(Deprecated) The text elements inside the <a>-tag.	String
nodeValue	(Deprecated) The text elements inside the <a>-tag.	String

StyleFields

This object is a FreeMarker “hash” container containing the field specifications for a given presentation style. The object can be accessed by field name, as defined in the presentation style defintion.

Property	Description	Type
`<name>`	The style field object for a given field. The field name refers to “reference name” from the presentation style definition. Note that the StyleField may also be reached through the “style” property on the DataField objects.	StyleField

StyleField

This object contains the properties defined on a single field in a presentation style.

The term “next sorting” is used to indicate that the property adapts to current state: If the field is not sorted, it returns a value suitable for ascending sort; if the field already has ascending sort, it returns a value suitable for descending sort; finally, if it already has descending sort, it returns a value suitable for ascending sort.

Property	Description	Type
label	The label value of the presentation style field.	String

externalKey	The external_key specified for the attribute behind the field.	String
isSortActive	Indicator to whether any sort is active for the field	Boolean
isSortActiveAscending	Indicator to whether ascending sort is active for the field	Boolean
isSortActiveDescending	Indicator to whether descending sort is active for the field	Boolean
sortAction	A clickable action for “next sorting”.	HtmlLink
sortActionAscending	A clickable action for ascending sorting	HtmlLink
sortActionDescending	A clickable action for descending sorting	HtmlLink
sortIconActive	HTML image tag displaying icon for currently active sort, if any.	String
sortIconAscending	HTLM image tag displaying either active or passive icon for ascending sort.	String
sortIconDescending	HTLM image tag displaying either active or passive icon for descending sort.	String
sortLabel	Deprecated, use “sortAction” instead.	HtmlLink
sortIconAsc	Deprecated, use “sortIconAscending” instead.	String
sortIconDesc	Deprecated, use “sortIconDescending” instead.	String

Logic

This object contains the set of fields defined in the presentation style

Property	Description	Type
languageId	The languageId of the viewer	String
noRecordsFound	The text to display if the viewer does not return any rows.	String
currentDate	Deprecated; use the value from the top level ContextModel instead (${context.date})	Date

Navigation

This object contains information supporting navigation in the viewer.

Property	Description	Type
pageSize	The number of rows defined to show on each page.	Number
firstRow	Index of the first row.	Number
lastRow	Index of the last row in the current data set (current page).	Number
totalRows	Total number of rows available to the viewer, even after navigation, if available.	Number

startRow	The first row requested (even though it may not actually exist).	Number
maxRow	Max number of rows to return by the viewer.	Number
isEmpty	True if there is no navigation bar.	Boolean
text	HTML of the entire navigation bar	String
range	The current navigation range (#-# of #)	String
first	Link to the first page of the viewer	NavigationLink
previous	Link to the previous page of the viewer	NavigationLink
next	Link to the next page of the viewer	NavigationLink
last	Link to the last page of the viewer	NavigationLink
goto	One or more links to specific row numbers	GotoModel
showMore	A link to the “show more” target defined for the viewer.	HtmlLink
moreLink	This property is deprecated; use “showMore” instead.	String

GotoModel

This object is a method model that lets you generate links to specific row numbers. As the object is a method, you call it using a method syntax as shown below.

Signature	Description	Type
(start)	Generate a single link	NavigationLink
(start, pagesize)	Generate a single link	NavigationLink
(start, interval, count)	Generate a list of links	Sequence of NavigationLink

To generate a simple link, use the syntax with a single argument:

Argument	Description	Type
start	Row number to generate link to	Number

To generate a link that both navigates to a specific row **and** sets the page size, use the syntax with two arguments:

Argument	Description	Type
start	Row number to generate link to	Number
pagesize	Number of rows per page	Number

To generate a list of links, use the syntax with three arguments. The method will return a list of links, useful for generating navigating bars with many links, for many pages:

Argument	Description	Type
start	Row number to generate link to	Number
interval	Interval between links	Number
count	Number of links to generate	Number

The list method will not return any illegal links. If you request goto(50,10,10), you are asking for 10 links to rows 50, 60, 70 ... 140. However, the method will not return links beyond the last record, so if the viewer has only 65 records, you would get two links only, to rows 50 and 60.

The list method will also allow you to generate links “backwards”, using a negative interval. goto(50,-10,10) will generate ten links 10 record intervals, up to row 50. Since it will not generate illegal links, this will in fact return five links: 10,20,30,40 and 50.

Examples:

```
<!-- To generate a single link to row number 50 -->
${viewer.navigation.goto(50)}

<!-- To generate five links before "this row", then "this row", then five
 links after -->
[#list viewer.navigation.goto(viewer.navigation.firstRow-10, -10, 5) as
 link]${link}[/#list]
${viewer.navigation.goto(viewer.navigation.firstRow)}
[#list viewer.navigation.goto(viewer.navigation.firstRow-10, +10, 5) as
 link]${link}[/#list]
```

HtmlLink

This object represents a single html link.

When used directly, the HtmlLink returns a string with the full link (<a>...);

Property	Description	Type
label	Localized text to use for the link text	String
url	The URL of the action, used as the “href” attribute of the <a>-tag.	String
onclick	The script to be used for the “onclick” attribute for an <a>-tag	String
script	The script to be used for the “onclick” attribute for an <a>-tag	String
target	The name of the html target (window) for an <a>-tag	String
<default>	The complete link element ((<a>...)	String

NavigationLink

This object represents a single navigation link.

When used directly, the NavigationLink returns a string with the full link (<a>...);

Property	Description	Type
label	Localized text to use for the link text	String
url	The URL of the action, used as the “href” attribute of the <a>-tag.	String
onclick	The script to be used for the “onclick” attribute for an <a>-tag	String
startRow	The first row to be displayed if the link is followed.	String
isValid	A value indicating whether a link is valid or not.	Boolean
<default>	The complete link element ((<a>...)	String

An invalid link has a start row outside the legal range. Both previous, next and last can return in invalid links, for example if you try to refer to the “previous” value when already on the first page. An invalid link has startRow set to null, url and onclick set to null, while the label contains the proper value.

8. Content search

The Content Search freemarker model is an extension of the Content Viewer, with an extra object available inside the ViewerModel.

Global objects

These are the global (top level) objects available when using FreeMarker from a Search Viewer.

Object	Description	Type
viewer	This object contains information about the viewer, its presentation style and its data.	ViewerModel
context	This object contains information about the execution context.	ContextModel
component	This object contains information about the executing component.	ComponentModel

ViewerModel

This object contains information about the Search Viewer itself, its definition and data. With the exception of the “search” property, all the others are documented along with the Content Viewer models.

Property	Description	Type
data	The data set of the content viewer.	DataModel
style	The presentation style of the content viewer.	StyleModel
logic	Logic information about the content viewer.	Logic
navigation	Navigational information about the content viewer.	Navigation
param	URL-parameters	ParameterModel
strings	Available strings for the viewer. This TextMapModel supports locations “style”, “viewer” and “domain”.	TextMapModel
search	Available search model information	SearchModel
`<default>`	When used directly, this object evaluates to the value of the “data” property.	

SearchModel

This object contains information about the Search Viewer itself, its definition and data.

Property	Description	Type
form	The search form, as HTML	String
term	The search term, as a string	String

9. XML viewer

Overview

The figure below gives a quick overview of the models available to the XMLViewer.

- xmlviewer
- xmlurl
- param
  - username
  - pageurl
  - referencepath
  - servername
  - serverport
  - guid
  - <name>
- context
  - user
    - id
    - username
    - token
      - name
      - value
  - date
  - time
  - datetime
  - millis

Global objects

These are the global (top level) objects available when using FreeMarker from a XMLViewer.

Object	Description	Type
xmlviewer	This object contains the loaded XML	XML
xmlurl	This object contains the URL that the XMl was loaded from	String
param	URL-parameters	ParameterModel
context	This object contains information about the execution context.	ContextModel
viewer	This property contains information about the viewer.	ViewerModel

ViewerModel

This object contains information about the XMLViewer.

Property	Description	Type
strings	Available strings for the viewer. This TextMapModel supports locations “template” and “domain”.	TextMapModel

10. Dimension viewer

Overview

The figure below gives a quick overview of the models available when rendering a Dimension Viewer using a template.

Global objects

These are the global (top level) objects available when using FreeMarker in a Dimension Viewer.

Object	Description	Type
component	This object contains information about the executing component.	ComponentModel
context	This property contains information about the execution context.	ContextModel
nodes	This property contains the top level nodes in the viewer.	DimensionListModel
viewer	This property contains information about the viewer.	ViewerModel

ViewerModel

This object contains information about the DimensionViewer.

Property	Description	Type
strings	Available strings for the viewer. This TextMapModel supports locations “template” and “domain”.	TextMapModel

DimensionListModel

This object contains the top level nodes in the viewer, and is really just a collection of DimensionModel objects.

The object is a freemarker sequence model, so you can access the nodes directly using `${nodes[index]}`, or you can traverse it using the `...[/#list]` syntax.

The object is also a freemarker node, so you can also use the built-ins ?children, ?parent, ?root, ?ancestors, ?node_name, ?node_type and ?node_namespace. The ?node_name and ?node_type properties are both “DimensionList”.

DimensionModel

This object contains information about a single dimension item.

The object is a freemarker node, so you can also use the built-ins ?children, ?parent, ?root, ?ancestors, ?node_name, ?node_type and ?node_namespace.

Property	Description	Type
objectId	The object ID of the node	Number
objectGuid	The object guid of the node	String
level	The level of the node	Number
label	The text of the node	String
isActive	An indicator of whether this node is the “active” one.	Boolean

hasActiveChild	An indicator of whether a sub-node of this one is the “active” one.	Boolean
url	The url of the menu item destination, typically used in the href-attribute of an <a>-tag.	String
action	The action specification for the dimension	DimensionActionModel
dimensionTypes	List of dimensions types applicable to this node	List of ObjectReferenceModel
?children	List of children nodes	List of DimensionModel
?parent	Parent node	DimensionModel
?root	Root node	DimensionModel
?ancestors	List of ancestors	List of DimensionModel
?node_namespace	There is no namespace; it always returns null.	Null
?node_type	Always the string “Dimension”	String
?node_name	Always the string “Dimension”	String

DimensionActionModel

This object contains information about a single dimension action, typically a URL link.

Property	Description	Type
label	The label value of the action	String
url	The URL of the action, used as the “href” attribute of the <a>-tag.	String
onclick	The value of the “onclick” attribute of the <a>-tag.	String
script	The value of the “onclick” attribute of the <a>-tag.	String
`<default>`	Formatted <a>-tag representing the action.	(Any)

11. Menu viewer

Overview

The figure below gives a quick overview of the models available when rendering a Menu Viewer using a template.

Global objects

These are the global (top level) objects available when using FreeMarker with a menu viewer.

Object	Description	Type
component	This object contains information about the executing component.	ComponentModel
context	This property contains information about the execution context.	ContextModel
nodes	This property contains the top level nodes in the viewer.	MenuItemListMode
viewer	This property contains information about the viewer.	ViewerModel

ViewerModel

This object contains information about the MenuViewer.

Property	Description	Type
portletTitle	The title of the portlet	String
strings	Available strings for the viewer. This TextMapModel supports locations “template” and “domain”.	TextMapModel

MenuItemListModel

This object contains the top level nodes in the viewer, and is really just a collection of MenuItem objects.

The object is a freemarker sequence model, so you can access the nodes directly using `${nodes[index]}`, or you can traverse it using the `...[/#list]` syntax.

The object is also a freemarker node, so you can also use the built-ins ?children, ?parent, ?root, ?ancestors, ?node_name, ?node_type and ?node_namespace. The ?node_name and ?node_type properties are both “MenuItemList”.

MenuItemModel

This object contains information about a single menu item.

The object is a freemarker node, so you can also use the built-ins ?children, ?parent, ?root, ?ancestors, ?node_name, ?node_type and ?node_namespace.

Property	Description	Type
objectId	The ID of the node	Number
objectGuid	The guid of the node	String
level	The level of the node	Number
label	The text of the node	String

isActive	An indicator of whether this node is the “active” one.	Boolean
hasActiveChild	An indicator of whether a sub-node of this one is the “active” one.	Boolean
url	The url of the menu item destination, typically used in the href-attribute of an <a>-tag.	String
action	The action specification for the menu item	DimensionActionModel
tooltip	The tooltip text of the menu item	String
cssClass	The text of the css information specified on the menu item	String
image	The name/path of the image specified on the menu item. Often, styling based on cssClass is a better choice.	String
imageRollover	The name/path of the rollover image specified on the menu item. Often, styling based on cssClass is a better choice.	String
?children	List of children nodes	List of MenuItemModel
?parent	Parent node	MenuItemModel
?root	Root node	MenuItemModel
?ancestors	List of ancestors	List of MenuItemModel
?node_namespace	There is no namespace; it always returns null.	Null
?node_type	Always the string “MenuItem”	String
?node_name	Always the string “MenuItem”	String

MenuItemActionModel

This object contains information about a single menu item action, typically a URL link.

Property	Description	Type
label	The label value of the action	String
url	The URL of the action, used as the “href” attribute of the <a>-tag.	String
onclick	The value of the “onclick” attribute of the <a>-tag.	String
script	The value of the “onclick” attribute of the <a>-tag.	String
`<default>`	Formatted <a>-tag representing the action.	(Any)

12. HTML Viewer

Overview

The figure below gives a quick overview of the models available when rendering a HTML Viewer portlet.

Global objects

These are the global (top level) objects available when using FreeMarker with a HTML Item Viewer.

Object	Description	Type
component	This object contains information about the executing component.	ComponentModel
context	This property contains information about the execution context.	ContextModel

13. BPEL Task form

Overview

The figure below gives a quick overview of the models available to the BPEL Task form.

- form
  - javascript
  - taskFields
  - param
  - claimurl
  - claimlink
  - needsClaim
  - notClaimable
  - claimedByUser
  - claimedByOther
  - claimedBy
  - username
- xmlform
  - ikbform
    - formcancel
    - formbegin
    - attributes
      - attribute
        - `<name>`
    - submit
      - apply
      - save
      - reset
- definition
  - outcomes
    - `<name>`
  - taskDefinitionId
  - title
  - priority

Global objects

These are the global (top level) objects available when using FreeMarker from a task form.

Object	Description	Type
form	This object contains the form information	TaskFormModel
xmlform	This object contains the URL that the XMl was loaded from	XML NodeModel
definition	Form defintion	DefinitionModel

TaskFormModel

This object contains information about the form.

Property	Description	Type
javascript	Javascripts that the form requires	String
taskFields	HTML containing required form elements for the form	String
param	Parameter information	ParameterModel
claimurl	URL to action for claiming a task	String

claimlink	Full HTML for <a> tag to claim a task	String
needsClaim	Boolean value indicating whether a value needs to be claimed	Boolean
notClaimable	Boolean value indicating whether a value can be claimed	Boolean
claimedByUser	Boolean value indicating whether this task has been claimed by this user	Boolean
claimedByOther	Boolean value indicating whether this task has been claimed by this user	Boolean
claimedBy	Username for user that has claimed task	String
username	Username for logged-on user	String

XMLFormModel

This object contains the form elements themselves, in an XML-structure easily accessible to FreeMarker's XML-functions.

Property	Description	Type
formcancel	HTML to generate a form that is required for the cancel action	String
formbegin	HTML containing required form elements for the form	String
attributes	This object contains all the attributes defined in the form.	AttributesModel
submit	This object contains all the submit actions in the form	SubmitModel

AttributesModel

This object, in fact an XML-based NodeModel, contains information on all the attributes in a form.

Property	Description	Type
attribute	Liste of all attributes	ListModel
<name>	A given attribute	AttributeModel

AttributeModel

This object, in fact an XML-based NodeModel, contains information about a single attribute in a form.

Property	Description	Type
label	Label for the attribute.	String
value	HTML to generate the input field for the attribute.	String

SubmitModel

This object, in fact an XML-based NodeModel, contains information about the normal submit-actions in a form.

Property	Description	Type
apply	HTLM to generate an apply-button	String

save	HTML to generate a save-button	String
reset	HTML to generate a reset-button	String

Definition

This object, in fact an XML-based NodeModel, contains information about the process task definition.

Property	Description	Type
outcomes	List of possible outcomes	ListModel
taskDefinitionId	ID for the task definition	String
title	Task title	String
priority	Task priority	Number

14. BPEL Task viewer

Overview

The figure below gives a quick overview of the models available to the BPEL Task viewer.

- wf
  - data
    - size
    - taskurl
    - claimurl
    - claimlink
    - needsClaim
    - notClaimable
    - claimedByUser
    - claimedByOther
    - claimedBy
    - username
    - `<name>`
      - text
      - number
      - date
      - link
      - fullname
      - `<object>`
  - style
    - `<name>`
      - label
  - param
  - strings
    - `<name>`

Global objects

These are the global (top level) objects available when using FreeMarker from a task viewer.

Object	Description	Type
wf	This object contains the task information	TaskModel

TaskModel

This object contains information about the task

Property	Description	Type
data	Data objects for a task	TaskDataModel
style	Style elements for a task	TaskStyle
param	URL parameter information for the viewer	ParameterModel

TaskDataModel

This object contains information about the data objects for a task.

Property	Description	Type
size	Number of tasks in list	Number
taskurl	URL to a page for viewing or editing a task	String

claimurl	URL to action for claiming a task	String
claimlink	Full HTML-link (<a>) for claming a task	String
needsClaim	Boolean value indicating whether a value needs to be claimed	Boolean
notClaimable	Boolean value indicating whether a value can be claimed	Boolean
claimedByUser	Boolean value indicating whether this task has been claimed by this user	Boolean
claimedByOther	Boolean value indicating whether this task has been claimed by this user	Boolean
claimedBy	Username for user that has claimed task	String
username	Username for user currently logged on	String
<name>	The presentation style field object for a given field. The field name refers to “reference name” from the presentation style definition.	TaskFieldModel

TaskFieldModel

This object contains a single field from a task viewer.

Property	Description	Type
text	The formatted text value of the field	String
number	The actual number value of the field, if applicable	Number
date	The date value of the field, if applicable	Date
link	Full HTML-link (<a>) to the task form, using the field value as the link text	String
fullname	The full name for a user, applicable only when the field itself contains the username of the user	String
<default>	Field value; the type depends on the field type	(Any)

TaskStyle

This object contains style information for a task viewer.

Property	Description	Type
<name>	The style field object for a given field. The field name refers to “reference name” from the presentation style definition.	TaskStyleField

TaskStyleField

This object contains style information for a task viewer field.

Property	Description	Type
label	The label value of the presentation style field	String

15. BPEL Process viewer

Overview

The figure below gives a quick overview of the models available to the ProcessViewer.

- process
  - list
  - node
  - hash
  - dimension

Global objects

These are the global (top level) objects available when using FreeMarker from a process viewer.

Object	Description	Type
process	This object contains the process information	ProcessModel

ProcessModel

This object contains information about the task.

Property	Description	Type
list	List of activities for the process	ListModel
node	Tree with activities for the process	XML NodeModel
hash	Hash model for activities for the process	HashModel
dimension	Tree of dimensions related to activities for the process	XML NodeModel

16. Activiti Form viewer

The Activiti Form viewer is the component used to render forms related to the Activiti process engine, related to either starting new processes or completing manual tasks.

A few nice-to-know items about the Activiti Form viewer:

- The form object is created the first time it is rendered. On repeated renderings (after submits, etc), the same form object is reused, with all the information intact (even if information is not rendered every time).
- Form components are “bound” to a data model, where they get their values from and store their values to. However, until a form is submitted or saved, values in the form component are independent from values in the data models.

Overview

The figure below gives a quick overview of the models available to the Activiti FormViewer:

- form
  - components
  - currentCommand
  - commonScripts
  - form
  - input
  - label
  - button
  - message
  - row
  - rows
  - debugString

FormModel

This object contains information about the form data and form components, as well as directives used to render the form.

Property	Description	Type
currentCommand	The current command of this invocation	HashModel
form	Directive used to print the html code for the form	FormDirective
input	Directive used to create and render a UI component for the form	FormInputDirective
label	Directive used to render a label for a component of the form	FormLabelDirective
button	Directive used to render an action button	FormButtonDirective
message	Directive used to render a message for a component of the form	FormMessageDirective
row	Directive used to render a html row with label and input	FormRowDirective
rows	Directive used to render html rows for many data elements	FormRowsDirective
components	Hash with user inteface components of the form	FormComponentModel

FormDataModel

This object contains information about a single Activiti FormData object for the form, either a Task form or a Start form, depending on usage.

Property	Description	Type
type	Type of form data, either “TaskFormData” or “StartFormData”	String
processDefinitionId	ID of activiti process definition	String
processInstanceId	ID of the process instance. For a start form, this has value only after submitting the form.	String
formKey	Form key defined on the activiti form	String
task	Wrapped object of the underlying task. Valid only for Task forms. For detailed information, see the FreeMarker documentation of the Task class	String
items	Map of items, with all values as strings.	HashModel

FormComponentModel

This object, accessible through `${form.components[_componentname_]}` (e.g `form.components["task.description"]` to access the component named “task.description”), contains information about a single form component. Note that the component models are only available after the component has been created through a `form.input` reference during the initial rendering; this means that the model is always available after a redisplay of the form (e.g. if validation fails, or if you use a command which does not submit data), but that the first time you display the form, the component does not exist prior to the `form.input` reference.

Property	Description	Type
name	Name of the component	String
bindExpression	Bind expression of the component	String
value	Value of the component	(Any)

Example:

```
<!-- Bind and value will be empty on initial rendering -->
Description: Bind=${form.components["task.description].bindExpression!}:
 Value=${form.components["task.description].value!}<br />

[@form.form debug="true"]
   ...
   [@form.input name="task.description" /]                  <!-- Creates the
 component during initial rendering -->
   ...
[/@form.form]

<!-- Bind will not be empty during initial rendering; value might be (depends
 on Activiti form defintition -->
Description: Bind=${form.components["task.description].bindExpression!}:
 Value=${form.components["task.description].value!}<br />
```

FormDirective

This directive is used to render a `<form>` html tag. The directive must be closed at the end.

Parameter	Description	Type
debug	Set to “true” to render debug information at end of form. Optional .	String

Example:

```
[@form.form debug="true"]
...
[/@form.form]
```

FormInputDirective

This directive is used to create a form component and render an input field for a data model item. The directive will automatically adapt its defaults according to the underlying data object’s semantics.

Parameter	Description	Type
name	Name you want to use for component; this will also be the “name” attribute in the html form.	String
bind	Path to object you want to load data from and store data to, such as “task.description”. If not present, it will default to the “name” attribute.	String
type	Type of input field, according to table below. Default value will depend on type of the bound object	String
attribute	External key of iKnowbBase attribute used to determine field type	String
`<default>`	Other parameters are rendered as they are, allowing any html attribute to be used and rendered.	String

The input fields can take many different forms, depending on its type. The type is initially decided from the underlying data field, such that a numeric field will render as a number field. You can also specify a type manually to override this; this is particularly useful when a single field type has many possible renderings (such as a list of values, which can render as select or radio, and sometimes even as a popup).

Type	Description
text	Normal text input field
number	Input field of type number
textarea	Normal text area
tinymce	Text area with tinymce editor
date	Text field with date popup and date format
datetime-local	Text field with datetime popup and format
hidden	Hidden input field
checkbox	Check box to toggle a boolean value on or off

radio	Radio list (buttons), with values popuplated from underlying data field
select	Select list (drop down), with values populated from underlying data field.
popup	Popup button, with values populated from underlying data field.

Examples:

```
<!-- Default input field -->
[@form.input name="task.title" /]

<!-- Text area and tinymce editor -->
[@form.input name="description" bind="task.description" type="textarea" /]
[@form.input name="content"     bind="task.content" type="tinymce"
 tinymce="small"/]

<!-- Renders a dimension navigator, a drop-down to select dimensions and a
 user popup to select user, based on attribute definitions -->
[@form.input name="dimNavPopup"  bind="task.organization"
 attribute="MY_ORG_DIM_ATTRIBUTE" /]
[@form.input name="dimNavSelect" bind="task.brand"
 attribute="MY_BRAND_DIM_ATTRIBUTE" /]
[@form.input name="userPopup"    bind="task.manager"
 attribute="MY_USER_ATTRIBUTE" /]
```

FormLabelDirective

This directive is used to render a label for a data model item, using the label defined in the underlying data model. For example, an Activiti Task item will have an attached label that you can render using this directive.

Parameter	Description	Type
bind	Path to object you want to render label for	String

Example:

```
[@form.label bind="task.description" /]
```

FormButtonDirective

This directive is used to render an action button for the form.

Parameter	Description	Type
name	Name of button to create. Required.	String
action	Action you want the button to perform. Use “submit” to submit form data; leave out to just reload form. Optional.	String

Examples:

```
<!-- Simple save-action -->
[@form.action name="submit" action="submit"]Complete task[/@form.action]

<!-- Three buttons, typically used on page 3 in a wizard. Use
 ${form.currentCommand.name} to decide which page to display -->
[@form.action name="page2"]Previous[/@form.action]
```

```
[@form.action name="page2"]Next[/@form.action]
[@form.action name="submit" action="submit"]Complete task[/@form.action]
```

FormMessageDirective

This directive is used to render a message for a component of the form. Messages are typically the result of validation failures (for most input fields), but are also used to store errors occuring during action processing.

Examples:

```
<!-- A typical scenario rendering a label, an input field and an error
 message -->
<tr>
<td>[@form.label bind="task.valuelist" /]</td>
<td>[@form.input name="task.valuelist"
 attribute="SYSTEST_ATTR_VALUELIST" /]</td>
<td>[@form.message for="task.valuelist" class="validationerror" /]</td>
</tr>

<!-- Renders an error message for a "submit" button -->
[@form.action name="submit" action="submit"]Complete task[/@form.action]
[@form.message for="submit" class="validationerror" /]
```

17. Page

Overview

The figure below gives a quick overview of the models available when rendering a Page using a template.

- page
  - resources
    - head
    - body
  - regions
    - <name>
  - clientPage
- strings
  - <name>
    - template
    - page
    - domain
- context
  - user
    - id
    - username
    - token
      - name
      - value
  - date
  - time
  - datetime
  - millis
  - request
    - pageurl
    - servername
    - serverport

Global objects

These are the global (top level) objects available when using FreeMarker in a PageViewer, or directly from the PageEngine.

Object	Description	Type
page	This property contains information about the page itself.	PageModel
context	This property contains information about the execution context.	ContextModel

PageModel

This object contains information about the page.

Property	Description	Type(s)
resources	Resources that need to be included in the final HTML-page.	ResourcesModel
regions	Page regions (as defined by the components, and not by the template).	RegionMacro, RegionModel

clientPage	Content of the clientPage, if there is a client page for this page execution.	String
strings	Available strings for the page. This TextMapModel supports locations “template”, “page” and “domain”.	TextMapModel

ResourcesModel

This object contains information about resources that need to be included in the final HTML-page, typically page title, scripts and css-references.

The object can be used as is, producing all the relevant resources:

```
<head>
    ${page.resources}
</head>
```

Alternatively, you can generate head and body properties separately. The head-property contains references that must be in the <head> part of the HTML-page, while the body-property contains references that may occur at the end of the <body> part:

```
<html>
<head>
    ${page.resources.head}
</head>
<body>
   ...
    ${page.resources.body}
</body>
```

Property	Description	Type
head	HTML for resources to generate in <head> of HTML-page	String
body	HTML for resources to generate in <body> of HTML-page	String
`<default>`	All resources at once, typically for use in <head> of HTML-page	String

RegionsModel

This model contains all the regions declared in the page. The model supports several of the basic FreeMarker model types:

- It is a TemplateHashModel, supporting the properties described in the table below.
- It is a TemplateHashModelEx, supporting the built-ins `?keys` and `?values`.
- It is a TemplateSequenceModel containing RegionModel, supporting index-based lookups (`page.regions[index]`) and iteration (`...[/#list]`).
- It is a TemplateNodeModel (where the children is the set of RegionModel), supporting node-based operations.

In addition to exposing the properties described in the table below, the model is also a freemarker sequence containing RegionModel; these can then be accessed using the syntax `page.regions[index]`.
The model is exposed as a sequence of RegionModel, as well
as a hash of RegionModel indexed by name, and a freemarker NodeModel which can be used for recursion.

Property	Description	Type

<name>	A single region	RegionModel
[index]	A single region, by number	RegionModel
?size	Number of regions on page	Number
?keys	Sequence of region names	List of String
?values	Sequence of regions	List of RegionModel
?children	List of children nodes	List of RegionModel
?parent	Parent node	Null
?root	Root node	Null
?ancestors	List of ancestors	List
?node_namespace	There is no namespace; it always returns null.	Null
?node_type	Always the string “Regions”	String
?node_name	Always the string “Regions”	String

RegionModel

This model contains all the portlets declared for a given region. The model supports several of the basic FreeMarker model types:

- It is a String (ScalarModel), returning the HTML for the entire region
- It is a TemplateHashModel, supporting the properties described in the table below.
- It is a TemplateSequenceModel containing PageComponentModel, supporting index-based lookups (`page.regions[index]`) and iteration (`...[/#list]`).
- It is a TemplateNodeModel (where the children is the set of PageComponentModel), supporting node-based operations.
- It is a TemplateMacro, used to print an entire region with or without a HTML-based container (`<div>...</div>`)

Property	Description	Type
name	Name of the region itself	String
html	HTML for the region content	String
decorated	HTML for the region content, always decorated	String
undecorated	HTML for the region content, never decorated	String
<default>	HTML for the entire region	String
[index]	A single page component, by number	PageComponentModel
?size	Number of regions on page	Number
?children	List of children nodes	List of PageComponentModel
?parent	Parent node	Null
?root	Root node	Null
?ancestors	List of ancestors	List
?node_namespace	There is no namespace; it always returns null.	Null
?node_type	Always the string “Region”	String
?node_name	Always the string “Region”	String

RegionModel is also a macro directive which can beused to print content for a region on a page, using the following syntaxes:

Using “.decorated” will return HTML-content where both the region itself and all containing portlets will be decorated with surrounding -tags, while using “.undecorated” will return HTML-content where

neither the region nor the containing portlets will be decorated. In general, use ".undecorated" for content rendered where the -tag will be illegal, such as inside the document <head>, or when rendering non-HTML content (JSON, XML, etc).

```
<@page.regions region="north" />
<@page.regions region="north" container="true" />
<@page.regions region="north" container="false" />
```

Parameter	Description	Type
region	Name of region that you want to print content for	String
container	Flag indicating whether you want to render a surrounding container or not. The default value is "true"	Boolean

Specifying "container=true" for the macro makes the page engine surround every component in the region with an HTML-container(...). This container is required for AJAX-based operations to work, and is recommended for most content. On the other hand, for content that is not HTML-based, or for content that is to be rendered outside the <body> of a HTML-page, the container may produce illegal output, and must be switched off.

PageComponentModel

This model contains a given page component (portlet). The model supports several of the basic FreeMarker model types:

- It is a String (ScalarModel), returning the HTML for the entire page component.
- It is a TemplateHashModel, supporting the properties described in the table below.
- It is a TemplateNodeModel (where the children is the set of PageComponentModel), supporting node-based operations.
- It is a TemplateMacro, used to print an entire region with or without a HTML-based container (...)

Property	Description	Type
name	Name of the page component itself	String
markupId	Markup ID for the page component	String
label	Printable and human readable label; often the portlet title	String
<default>	HTML for the entire region	String
?children	List of children nodes	Null
?parent	Parent node	Null
?root	Root node	Null
?ancestors	List of ancestors	List
?node_namespace	There is no namespace; it always returns null.	Null
?node_type	Always the string "PageComponent"	String
?node_name	Always the string "PageComponent"	String

18. Javascript library

iKnowBase provides a javascript-library in the client page. For the most part, the content of this is private for the various components. However, for components in the page engine there is one feature that might be useful.

The objects, methods and properties in the iKnowBase javascript library are generally not intended for manual use, and they will change between versions. However, as far as possible, the items specified in this document will remain backwards compatible.

Global objects

These are the global (top level) objects available to HTML clients, provided automatically by iKnowBase.

Object	Description	Type
iKnowBase	The iKnowBase javascript root object	iKnowBase

iKnowBase root object

This object contains functions and properties relating to the entire iKnowBase application.

Property	Description	Type(s)
PageEngine	This property contains the PageEngine object, with methods to refresh page components.	PageEngine
busyIndicator	This method will run a given function and show a busy indicator while the function runs.	Function
notify	This method will show a notification to the end user.	Function

busyIndicator

The busyIndicator method on the iKnowBase-object will run the given function and show a busy indicator while the function runs. It can apply to either a whole page or a given component.

Parameters

Parameter	Description	Type
options	Can be either a function to be started by the busy indicator, or a hash (object) with properties, as described in the table below.	Function/Object

The hash parameter supports the following properties:

Properties	Description	Type
autoClose	If `true` the busy indicator will close automatically, else closing the busy indicator must be handled manually. Defaults to: `true`.	Boolean
parent	If specified the busy indicator will display for the given component, else it will display for the body. It can be specified as a jQuery	String/Object

	selector, element, HTML string, or jQuery object.	
start	Specifies the function to be started by the busy indicator. **Note**: This is a required property.	Function
text	Specifies the text to display with the busy indicator.	String

Methods

Method	Description
close	Closes the busy indicator.

Example

The following function will show the busy indicator while loading css and scripts. It is manually closed in the start-function's callback.

```
IKB.ensure = function(data, callback) {
    iKnowBase.busyIndicator({autoClose: false, start: function() {
        var busy = this;
        load($.makeArray(data.js), $.makeArray(data.css), data.wait,
 function() {
            busy.close();
            callback();
        });
    }});
}
```

notify

The notify method on the iKnowBase-object will show a notification to the user. It can be of type error, warning, or information. The notification will close when the user clicks on it.

Parameters

Parameter	Description	Type
options	A hash (object) with properties, as described in the table below.	Object

The hash parameter supports the following properties:

Properties	Description	Type
duration	Specifies the number of milliseconds to show the notification. It not specified, the notification will remain open until it is manually closed.	Number
text	Specifies the text/description.	String
title	Specifies the title.	String
type	Specifies the type of notification. Supported values: 'error', 'info', and 'warning'.	String

Methods

Method	Description
close	Closes the notification.

Example

The following code will show a notification of type information to the user. It will close automatically after 3 seconds.

```
iKnowBase.notify({
    type:'info',
   title:'Shows both info-icon, title, and text',
   text:'Will automatically close after 3 seconds',
   duration:3000
});
```

iKnowBase.PageEngine

This object contains functions and properties relating to the page engine.

Property	Description	Type(s)
reloadComponent	This method will reload a component on a page, optionally passing in a set of URL parameters to use during the reload.	Function

reloadComponent

The reloadComponent method on the PageEngine-object will reload the specified page component from the server, using a number of different settings. The basic syntax is "iKnowBase.PageEngine.reloadComponent (settings)", where *settings* is a javascript hash containing the required settings and parameters, but there are also two old-style supported syntaxes:

It is possible to pass a set of URL-parameters for use during the reload. Whether URL-parameters are required or not, and what various parameters mean, will depend entirely on the (user defined) configuration of the specified page component.

Syntax and parameters

The basic syntax is "iKnowBase.PageEngine.reloadComponent (settings)", where *settings* is a javascript hash containing the required settings and parameters. The settings object supports the following properties:

Parameter	Description	Type
pageComponent	The client-side ID of the component to reload, or a jQuery-object pointing to the component	String, Element or jQuery
busyIndicator	Whether to show a busy indicator during the reload. Defaults to false.	Boolean
refreshCache	Whether to refresh any current cached content, overriding cache timeouts. Defaults to false.	Boolean
updateHtml	Whether to update the html of the component, or just load data. Defaults to true.	Boolean
errorPopup	Whether to automatically display errors, or ignore them. Defaults to true.	Boolean
data	A hash (object) with URL-parameters to be used during reload	hash

The return value from the reloadComponent is a jQuery promise object (actually a jqXHR object) making it possible to attach callbacks using methods such as done(), fail() and always(). See the example below for more information.

Note: In order to use updateHtml, the containing page region must be rendered with decorations, as specified by the `page.regions` FreeMarker model.

The ID of a page component is automatically and computed by the page engine, using an algorithm that may change between versions. You should generate the component ID during component rendering, using the ComponentModel expression ${component.id}. However, you may also specify the pageComponent ID in the page components tab in iKnowBase Development Studio.

The following example reloads the value of a single component, using the parameter p_document_id=1234:

```
iKnowBase.PageEngine.reloadComponent ({
    pageComponent: '${component.id}',
    data: {
        p_document_id: 1234
    }
});
```

The following example adds the busy indicator during reload and ensures that the page component content is regenerated (and not loaded from cache):

```
iKnowBase.PageEngine.reloadComponent ({
    pageComponent: '${component.id}',
    busyIndicator: true,
    refreshCache:  true,
    data: {
        p_document_id: 1234
    }
});
```

The following example turns off automatic display of errors, and uses the callbacks *done* and *fail* for visual status notifications:

```
iKnowBase.PageEngine.reloadComponent ({
    pageComponent: id,
    errorPopup: false
}).done (function (data, textStatus, jqXHR) {
    jQuery("#" + id).css("background-color", "green");
}).fail (function (jqXHR, textStatus, errorThrown) {
    jQuery("#" + id).css("background-color", "red");
});
```

Finding the id of the page component

The ID of a page component is normally computed by the page engine, using an algorithm that may change between versions. You should generate the component ID during component rendering, using the FreeMarker expression ${component.id}. You may also specify a stable pageComponent ID in the page components tab in iKnowBase Development Studio, but you are then responsible for ensuring that there are no conflicts.

Use the following template in a ContentViewer to create a function that will reload the viewer with new data:

```
<script type="text/javascript">
    function showDocument (docid) {
        iKnowBase.PageEngine.reloadComponent ({
            pageComponent: '${component.id}',
            data: {
```

```
                p_document_id: docid
            }
        });
    }
</script>
<h2> ${viewer.data.title}</h2>
```

Deprecated syntax and parameters

reloadComponent still supports a less versatile syntax “iKnowBase.PageEngine.reloadComponent (pageComponent [, url] [, urlParameters]”. The traditional syntax has both url and urlParameters as optional function parameters, and it will automatically detect if they are present.

Parameter	Description	Type
pageComponent	The client-side ID of the component to reload, or a jQuery-object pointing to the component	String, Element or jQuery
url	URL to the page to reload from, possibly including URL-parameters	String
urlParameters	A hash (object) with additional URL-parameters to be used during reload	hash

Note that reloadComponent requires that the containing component has been rendered with decorations, as specified by the `page.regions` FreeMarker model.

iKnowBase.PageEngine.InstantContentCache

This object contains functions and properties relating to the InstantContentCache, i.e. the mechanism that enables clients to receive notification when content caches have been updated. (Note that for this to work, the content cache key must have “publish updates to instant” enabled).

Property	Description	Type(s)
start	This method will start listening for cache notifications, automatically reloading any component which is present on the page	Function

start

The reloadComponent method on the PageEngine.InstantContentCache-object will start listening for cache notifications. Before starting, it will check whether it has any page components with known cache keys; if there are none, the listening process witll not start. Whenever a notification is received, it will check whether a component with that particular cache key is known on the page; if it is, it will use PageEngine.reloadComponent to update the data.

Syntax and parameters

The basic syntax is “iKnowBase.PageEngine.InstantContentCache.start (settings)”, where *settings* is a javascript hash containing the required settings and parameters. The settings object is currently undocumented:

Parameter	Description	Type
reloadComponent	The function to be called to reload a component. Parameters to this function is a hash containing only the attribute pageComponent. If the function is not specified,	Function

	iKnowBase.PageEngine.reloadComponent will be used.

The following example will start listening to cache update notifications, using a default reloadComponent:

```
iKnowBase.PageEngine.InstantContentCache.start();
```

The following example will start listening to cache update notifications, telling the user that the page might need to be refreshed:

```
iKnowBase.PageEngine.InstantContentCache.start({
    reloadComponent: function (settings) {
        iKnowBase.notify ({ text:'Content has been updated. Please refresh
 the page.' });
    }
});
```

iKnowBase.Instant

The iKnowBase.Instant object contains functions related to the use of iKnowBase Instant, the component providing real time asynchronous messaging to iKnowBase. Some of the available options are in a sub-object named Atmosphere, so named to indicate that these rely on implementation details of the underlying Atmosphere framework.

For simple usage scenarios, the examples below and in the Development Guide may be sufficient; for more advanced usages you may want to consult and understand the underlying Atmosphere APIs.

Property	Description	Type(s)
Atmosphere.subscribe	Subscribes to a topic, and returns the Atmosphere channel.	Function
Atmosphere.publish	Publishes a message to an already subscribed channel.	Function
Atmosphere.unsubscribe	Unsubscribes from a channel already _subscribe_d to.	Function
Atmosphere.requestUserListUpdat	For userList topics: Request a userList response from the server. Same as subscription option subscribeUserListChanges, but used on demand.	Function
Atmosphere.requestUserList	For userList topics: Request a filtered userList response from the server.	Function
Atmosphere.requestSubscribeUse	For userList topics: enable or disable join/leave updates for the active connection. Same as subscription option subscribeUserListChanges, but used on demand.	Function
setDefaultOptions	Sets default options to be used for all future subscribe-calls.	Function

Atmosphere.subscribe

This method establishes a topic-specific channel between the client and a server. The method also sets up callbacks for various events.

Syntax

```
iKnowBase.Instant.Atmosphere.subscribe (options)
```

Parameter	Description	Type
options	Options to be used, a combination of iKnowBase- and Atmosphere specific options.	Hash

Options

The options are as follows:

Parameter	Description	Type
topicName	The full name of the topic you want to connect to.	String
servletURL	CORS only: URL to Instant servlet. Used in conjunction with enableXDR atmosphere option.	String
subscriptionOptions	iKnowBase-specific options that apply to the subscription. Optional.	Hash
atmosphereOptions	Atmosphere-specific options that apply to the subscription. Optional.	Hash

The subscriptionOptions are as follows:

Parameter	Description	Type
requestUserList	Request that the server provides a full list of subscribed users as soon as the client has connected. Optional.	Boolean
subscribeUserListChanges	Request that the server provides join and leave messages when users subscribe and unsubscribe to the topic. Optional.	Boolean

The atmoshpereOptions are better described in the Atmosphere documentation, but below are the most important.

Parameter	Description	Type
transport	Name of transport to use; Legal values are ‘websocket’ and ‘long-polling’; defaults to ‘long-polling’.	String
fallbackTransport	Name of transport to use if the preferred transport is unavailable (or fails); defaults to ‘long-polling’.	Hash
enableXDR	CORS only: Enable CORS communication. Used in conjunction with servletURL option.	String
onMessage	Callback for when a messsage is received. Receives a single “data” parameter, whose format depends on the topic setup.	Function
onOpen	Callbacks for when the channel is opened.	Function
onClose	Callbacks for when the channel is closed.	Function

onReconnect	Callback for when the client reconnects to the server.	Function
onReopen	Callback for when a re-connection successfully reconnected.	Function
onError	Callback for when the client discovers an error.	Function
onMessagePublished	Callback for when using polling and a response was sent back by the server.	Function
onTransportFailure	Callback for when the transport fails because it is not supported by the client or the server.	Function

Topic name and topic options

The topicName option consists of a topic base name and a set of optional topic options. The combination will uniquely identify a specific topic.

The topic base name is required to start with a “/” (forward slash) and can otherwise only contain alphanumeric characters [a-z|A-Z|0-9] and additional forward slashes. The name is case sensitive.

The topic options are added to the topic base name as URL parameters:

Parameter	Value	Description
messageFormat	TEXT	Use simple text message format when publishing and consuming on the topic. This is the default if messageFormat is not set.
messageFormat	IKB	Use IKB message format when publishing and consuming on the topic.
userList	true	Enable userList support. Requires messageFormat=IKB. Instant will keep a list of all connected users and support subscriptionOptions requestUserList and subscribeUserListChanges.

While the TEXT message format can be used to send any text information, the IKB message format has additional application support:

Publishers should send messages using a IKBRequestMessage:

Parameter	Description	Type
toUserName	The userName of a the recipient. The message will only be delivered to connections with that userName. Optional.	String
messageType	A message type identifier the clients can use to describe the message data payload or reason for message. Optional.	String
correlationId	A optional user specific identifier that can be used with server requests (iKnowBase.Instant.Atmosphere.request*). The identifier will be present in the corresponding response.	String

data	The data to be sent. Required.	String

Note: Publishers can also choose to send a simple text message to a topic with messageFormat=IKB. This message will be set as data and all optional fields will be empty.

Consumers will always receive messages as a IKBResponseMessage:

Parameter	Description	Type
fromUser	The UserReference object of the sender. Present if the user was authenticated, otherwise null.	Hash
toUserName	The userName of a the recipient. The recipient can see if this was a private message or a public message. Optional.	String
messageType	A message type identifier the clients can use to describe the message data payload or reason for message. Optional.	String
correlationId	A optional user specific identifier that can be used with server requests (iKnowBase.Instant.Atmosphere.re The identifier will be present in the corresponding response.	String
data	The data sent by the publisher.	String

Examples

This example establishes a connection to an instant topic, with a single onMessage-callback.

```
var channel = iKnowBase.Instant.Atmosphere.subscribe ({
    topicName: "/blog/entries",
    atmosphereOptions: {
        onMessage: function (data) { /* Do something with message */ }
    }
});
```

This example establishes a connection to an instant topic with specified options, including:

Option type	Option	Value
Topic	messageFormat	IKB
Topic	userList	true
Subscription	requestUserList	true
Subscription	subscribeUserListChanges	true
Atmosphere	transport	websocket
Atmosphere	onMessage	a callback function
Atmosphere	onError	a callback function

```
[#ftl]
[#if context.user.isLoggedOn]
 [#assign secureTokenAuth =  "'_ikbUserToken':" + "'" +
 context.user.token.value + "'"]
[#else]
 [#assign secureTokenAuth = '']
[/#if]
var channel = iKnowBase.Instant.Atmosphere.subscribe ({
    topicName: "/blog/entries?messageFormat=IKB&userList=true",
```

```
    subscriptionOptions: {
        requestUserList: true,
        subscribeUserListChanges: true,
        ${secureTokenAuth} // Authentication header from ikbViewer
    },
    atmosphereOptions: {
        transport: "websocket",
        onMessage: function (data) { /* Do something with message */ },
        onError: function (atmosphereResponse) { /* Do something with message
 */ }
    }
});
```

Atmosphere.publish

This method establishes a topic-specific channel between the client and a server. The method also sets up callbacks for various events.

Syntax

```
iKnowBase.Instant.Atmosphere.publish (channel, data)
```

Parameter	Description	Type
channel	Channel to publish message on, as returned from *subscribe()*.	Object
data	Data to publish on channel. Type depends on topic messageFormat; use hash for IKB and String for TEXT	String or Hash

Example

To send a text message on a channel with messageFormat=TEXT

```
iKnowBase.Instant.Atmosphere.publish (channel, "Everybody, there is cake in
 the reception!");
```

To send a direct message on a channel with messageFormat=IKB

```
iKnowBase.Instant.Atmosphere.publish (channel, {
    toUserName: "NARMSTRONG",
    messageType: "WALK_ON_MOON",
    data: "One giant leap, indeed!"
});
```

Atmosphere.unsubscribe

This method establishes a topic-specific channel between the client and a server. The method also sets up callbacks for various events.

Syntax

When called without parameters, the unsubscribe-function closes all open connections

```
iKnowBase.Instant.Atmosphere.unsubscribe ()
```

When called with a single channel parameters, the unsubscribe-function closes the connection to that channel.

```
iKnowBase.Instant.Atmosphere.unsubscribe (channel)
```

Parameter	Description	Type

channel	Channel to unsubscribe to, as returned from *subscribe()*.	Object

Atmosphere.requestUserListUpdate

Request a userList response from the server. Same as subscription option subscribeUserListChanges, but used on demand.

Syntax

```
iKnowBase.Instant.Atmosphere.requestUserListUpdate (channel, correlationId)
```

Parameter	Description	Type
channel	Channel for requesting the server information, as returned from *subscribe()*.	Object
correlationId	An optional user specific identified that will be present in the server response.	String

Result is an onMessage call containing IKBResponseMessage with messageType=IKB.USERLIST.RESPONSE and data=List<UserReference>.

Requirements

Active connection on a userList topic.

Example

To request a userList update:

```
iKnowBase.Instant.Atmosphere.requestUserListUpdate (channel,
 "MY_CORRELATION_ID");
```

onMessage response.responseBody:

```
{
   "toUserName":"ORCLADMIN",
   "fromUser":{
      "guid":null,
      "id":0,
      "username":"ikb$instant",
      "dn":null,
      "label":"iKnowBase Instant Server"
   },
   "messageType":"IKB.USERLIST.USERS",
   "correlationId":"MY_CORRELATION_ID",
   "data":[
      {
         "guid":"BBEA12EB56126795E040000A180038B7",
         "id":58466,
         "username":"KERMIT",
         "dn":null,
         "label":"kermit frog"
      },
      {
         "guid":"8582B1E8B4AA4BDA9B7E0B6422A1D1F7",
         "id":125,
         "username":"ORCLADMIN",
         "dn":"orcladmin",
         "label":"Administrator orcladmin"
      }
```

```
   ]
}
```

Atmosphere.requestUserList

Request a filtered userList response from the server. Typically used to check a subset of the result from subscribeUserListChanges.

Syntax

```
iKnowBase.Instant.Atmosphere.requestUserList (channel, correlationId,
 userNames)
```

Parameter	Description	Type
channel	Channel for requesting the server information, as returned from *subscribe()*.	Object
correlationId	An optional user specific identified that will be present in the server response.	String
userNames	An optional array or ; delimited String of usernames used as filter.	String

Result is an onMessage call containing IKBResponseMessage with messageType=IKB.USERLIST.RESPONSE and data=List<UserReference>.

Requirements

Active connection on a userList topic.

Example

To request a filtered userList:

```
// Example where a full userList would result in ORCLADMIN and KERMIT. Apply
 KERMIT as filter.
iKnowBase.Instant.Atmosphere.requestUserList (channel, "MY_CORRELATION_ID",
 "KERMIT");
```

onMessage response.responseBody:

```
{
   "toUserName":"ORCLADMIN",
   "fromUser":{
      "guid":null,
      "id":0,
      "username":"ikb$instant",
      "dn":null,
      "label":"iKnowBase Instant Server"
   },
   "messageType":"IKB.USERLIST.RESPONSE",
   "correlationId":"MY_CORRELATION_ID",
   "data":[
      {
         "guid":"BBEA12EB56126795E040000A180038B7",
         "id":58466,
         "username":"KERMIT",
         "dn":null,
         "label":"kermit frog"
      }
   ]
```

```
}
```

Atmosphere.requestSubscribeUserListChanges

Enable or disable join/leave updates for the active connection. Same as subscription option subscribeUserListChanges, but used on demand.

Syntax

```
iKnowBase.Instant.Atmosphere.requestSubscribeUserListChanges (channel,
 correlationId, newSubscribeUserListChangesState)
```

Parameter	Description	Type
channel	Channel for requesting the server information, as returned from *subscribe()*.	Object
correlationId	An optional user specific identified that will be present in the server response.	String
newSubscribeUserListChangesState	Boolean for if subscribeUserListChanges should be enabled(true) or disabled(false).	Boolean

Result is an onMessage call containing IKBResponseMessage with messageType=IKB.USERLIST.RESPONSE and data=List<UserReference>.

Requirements

Active connection on a userList topic.

Example

To enable subscribeUserListChanges:

```
iKnowBase.Instant.Atmosphere.requestSubscribeUserListChanges (channel,
 "MY_CORRELATION_ID", true);
```

onMessage response.responseBody:

```
{
   "toUserName":"ORCLADMIN",
   "fromUser":{
      "guid":null,
      "id":0,
      "username":"ikb$instant",
      "dn":null,
      "label":"iKnowBase Instant Server"
   },
   "messageType":"IKB.USERLIST.SUBSCRIBE.RESPONSE",
   "correlationId":"MY_CORRELATION_ID",
   "data":"Change request for SubscribeUserListChanges: old=false; new=true"
}
```

To disable subscribeUserListChanges:

```
iKnowBase.Instant.Atmosphere.requestSubscribeUserListChanges (channel,
 "MY_CORRELATION_ID", false);
```

onMessage response.responseBody:

```
{
   "toUserName":"ORCLADMIN",
```

```
   "fromUser":{
      "guid":null,
      "id":0,
      "username":"ikb$instant",
      "dn":null,
      "label":"iKnowBase Instant Server"
   },
   "messageType":"IKB.USERLIST.SUBSCRIBE.RESPONSE",
   "correlationId":"MY_CORRELATION_ID",
   "data":"Change request for SubscribeUserListChanges: old=true; new=false"
}
```

setDefaultOptions

This method sets default options to be used for all future subscribe-calls

Syntax

```
iKnowBase.Instant.setDefaultOptions (options)
```

When called with a single channel parameters, the unsubscribe-function closes the connection to that channel

```
iKnowBase.Instant.Atmosphere.unsubscribe (channel)
```

Parameter	Description	Type
options	Options to be used, a combination of iKnowBase- and Atmosphere specific options. See *Atmosphere.subscribe()* for details.	Hash

Complete example

```
functon displayMessageNotification (data) {
    iKnowBase.notify({
        title: "New message received",
        text: data
    });
}

var channel = iKnowBase.Instant.Atmosphere.subscribe ({
    topicName: "/blog/entries",
    atmosphereOptions: {
        transport: "websocket",
        onMessage: displayMessageNotification
    }
});
```

19. Content Server

The iKnowBase Content Server is the web endpoint for accessing document content. The content server supports both getting (downloading) and putting (uploading) content over the HTTP-protocol.

Download content

To download content, use the HTTP GET-method with a URL identifying a specific document. The format of the URL is a set of options, separated by slashes and/or commas, ending with a trailing (and unused) readable name:

```
/Content/id/option,option,option,.../name
/Content/id/option/option/option/.../name
```

You may use either one or both of the conventions: Separate the options by a forward slash, or use the comma, or a combination of both. Also, there is no restriction on the ordering of the options.

Option	Description
id	A number specifying the document ID of the document whose content you want to retrieve
document.id=number	Specifies the document ID of the document whose content you want to retrieve
document.guid=guid	Specifies the document guid of the document whose content you want to retrieve
document.externalKey=key	Specifies the external key of the document whose content you want to retrieve
docid=number	Specifies the document ID of the document whose content you want to retrieve
guid=guid	Specifies the document guid of the document whose content you want to retrieve
version=number	Specifies the version number of the document that you want to retrieve
attr=guid	Specifies the guid of a file attribute whose content you want to retrieve
cache=millis	This option specified that you want the server to set HTTP-headers that request the document to be cached, *if* the lastChangedDate of the document matches the given millis-value.
expires=seconds	Specifies that you want the server to set HTTP-headers that request the document to be cached for the specified number of seconds.
no-cache	Specifies that you want the server to set HTTP-headers that request the document not to be cached.
accept=text	Specifies that you want the content object converted to and returned as a pure text file. The server response will use the Mime-type ï¿½text/plain"
accept=html	Specifies that you want the content object converted to and returned as an HTML file. The server response will use the Mime-type ï¿½text/html"
accept=zip	Specifies that you want the content object zipped and returned as a zip file. Most useful along when downloading multiple files (see below).

content-disposition=inline	Specifies that you want the server to set the HTTP header ï¿½content-disposition: inline". This is a request to the browser that the document is opened inline, inside the browser window.
content-disposition=attachment	Specifies that you want the server to set the HTTP header ï¿½content-disposition: attachment". This is a request to the browser that the document is opened as an attachment, in a special application outside the browser window.
name	Any name you want in the URL. The content server does in fact not use this for anything, but it is useful for providing a human readable URL, and to provide an extension for the web browser.

```
/Content/1234/FlightPlan.doc
```

The URL above will download the content of document #1234. The part “FlightPlan.doc” is not used by the content server, but may help the user to better understand what the document is about.

```
/Content/document.id=1234/version=2/accept=html/FlightPlan.html
```

The URL above will download the content of document #1234, version 2, converted to HTML. Again, the part “FlightPlan.html” is not used by the content server, but may help the user to better understand what the document is about.

```
/Content/document.externalKey=KEY_CURRENT_FLIGHT_PLAN/accept=pdf/
FlightPlan.pdf
```

The URL above will download the content of a document with external key “KEY_CURRENT_FLIGHT_PLAN”, converted to PDF. Again, the part “FlightPlan.pdf” is not used by the content server, but may help the user to better understand what the document is about.

Download multiple files

You may use the Content Server to download multiple files at once. To do this, add a number of entry-parameters to the URL, where each parameter is a full specification for the Content Server. For example, use the following (one line, broken here for readability):

```
/Content/accept=zip/test.zip
?entry=/1111/mydocument.doc
&entry=/1111/accept=text/mydocument.text
&entry=/1111/accept=html/mydocument.html
&entry=/docid=2222/version=2/instructions.pdf
```

The URL above specifies the following:

- Download a zip-file (accept=zip)
- Include document ID 1111, named ï¿½mydocument.doc"
- Include document ID 1111, converted to text, named ï¿½mydocument.text"
- Include document ID 1111, converted to html, named ï¿½mydocument.text"
- Include document ID 2222, version 2, named ï¿½instructions.pdf"

Upload content

You may use the ContentServer to upload files, either as new documents or new content for existing documents.

Existing documents

To upload new content for existing documents, POST data to the Content Server, using the same URL that will server document content (shown above). You will need to be logged in for this to work.

New documents

Uploading new documents through the ContentServer works together with the UploadEvent mechanism: The Content Server receives the file, and then triggers an UploadEvent. The implementation of the UploadEvent will then handle the actual storage of the new document.

To upload new documents, POST data to the root of the Content Server. In addition to the files you upload, you may also pass a parameter “uploadParameters” that are forwareded to the UploadEvent; the content of this can be used to specify how the content should be handled.

Uploading content through the file server is most useful with new HTML5-mechanisms, where the following could be used to enable drag-and-drop of files into iKnowBase:

```
function startUpload(files, uploadParameters) {
   var xhr = new XMLHttpRequest();
   var data = new FormData();
   data.append("uploadParameters", uploadParameters);
   for (var i = 0; i < files.length; i++)
      data.append("file", files[i]);
   xhr.open("POST", "/private/content");
   xhr.send(data);
}
```

20. Page Engine

The iKnowBase Page Engine is the server component responsible for handling web requests from clients (web browsers, tablets, smart phones, etc). The Page Engine will find the proper page, and return it to the client.

General use

The basic use of the page engine is to parse the URL, find the page requested, build the page content, and return it to the client. The page engine is available both at the application root, and under the “/page” endpoint.

For example, if the iKnowBase ikbViewer application is mounted on “/ikbViewer”, both of the following requests will find the page “/about”, and return to the client:

```
/about
/page/about
```

Normally, you will use the root-based mountpoint (/about). However, there are some other services that have higher priority, such as the content server on /Content. Therefore, it is not possible to access a page named “/Content” using “/Content”, and you will have to use the specific endpoint “/page/Content”.

Language support

When a user first accesses the Page Engine, he will get a user session. One of the items stored in the session is the current language, where the initial value is decided from the domain configuration, the user settings or the user browser. However, the session language can be changed at runtime, valid only for that session.

For developers, the console available at /ikb$console provides a mechanism for changing the user language. This is readily available under the “developer” tab, and is very convenient for testing.

For application development, the language can also be changed using the URL-parameter “_ikbLanguageCode”, with the value set to one of the available language codes. Use the URL-parameter on any page-request; the following two examples will change the language to Norwegian or English:

```
/about?_ikbLanguageCode=n
/page/about?_ikbLanguageCode=us
```

Database trace functionality

The page engine (and really most of the iKnowBase infrastructure) uses a single database connection per request. It is possible to ask that all sql executed by that request should be traced in the Oracle database.

To enable tracing for a single request, use the optional URL parameter “_ikbRequestTracefile=trace_identifier”, as follows:

```
/page/frontpage?_ikbRequestTracefile=frontpage
```

Then, the database will create a trace file to the default trace location (which varies between database installations), using the specified trace identifier. For example, on an 11g installation this could be “/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_4851_frontpage.trc”.

Refresh content cache

Page components that are configured to use content caching will automatically deliver their content from cache, whenever the cache is present and valid (not expired). However, it is possible to ask that a single request should refresh all content, and not delivered anything from cache.

To enable cache refresh for a single request, use the optional URL parameter “_ikbRefreshContentCache=true”, as follows:

```
/page/frontpage?_ikbRefreshContentCache=true
```

You may also use the “refreshCache” option on the loadComponent javascript-api, which in fact adds a “_ikbRefreshContentCache” parameter under the hood:

```
iKnowBase.PageEngine.reloadComponent ({
    pageComponent: "ikb4",
    refreshCache: true
})
```

21. Form HTML Template support

When you chose to use html-based generation for a Content form or Search source, you must supply the HTML markup that is used when the form is rendered. In the markup you provide, you can use a number of iKnowBase-specific tags, which are replaced with appropriate markup and information at runtime.

For getting started, use the Create new template button on the HTML tab; this will produce a sample template you can modify as needed.

Overview of iKnowBase-specific tags

Tag	Description
<IKB:FORM>	Renders a HTML form-tag with the appropriate information.
<IKB:FORMNAME>	Renders the name of the HTML-form.
<IKB:LABEL>	Renders the label for a form field.
<IKB:INPUT>	Renders an input field for a form field.
<IKB:DIMENSION>	Renders a dimension selector for a form field.
<IKB:FAVORITE>	Renders a drop-down list of favorite dimensions for a form field.
<IKB:SPACE>	Renders a non-breaking space.
<IKB:CALENDAR>	Renders the calendar button for a form field.
<IKB:RADIO>	Renders a radio button for a form field.
<IKB:CHECK>	Renders a checkbox for a form field.
<IKB:SELECT>	Renders a select list for a form field.
<IKB:BUTTON>	Renders a button for the form.
<IKB:RELATED_OBJECT>	Renders a button for selecting ï¿½related objects" for a form field.
<IKB:RELATED_IMAGE>	Renders a button for selecting ï¿½related image" for a form field.
<IKB:CONDITIONAL>	Works as an if-test, conditionally rendering or skipping the content, depending on document state.
<IKB:FILE>	Renders a button to upload file content.
<IKB:FILELINK>	Renders the current filename of file content.
<IKB:FILEURL>	Renders the current link to file content.
<IKB:ATTACHMENT>	Renders a button for adding attachments to the document being edited.
<IKB:TEXTAREA>	Renders a textarea for a form field.
<IKB:ADDATTRIBUTE>	Renders a drop down list for adding more attributes to the form at run time.
<IKB:DISPLAY_ADD_ATTR>	Renders a list of dynamic attributes added to the form at run-time. Applicable to Search sources.
<IKB:PROMPT>	Renders the prompt defined for the form field.
<IKB:DISPLAY>	Renders a readable text value of a form field.
<IKB:CONDITIONAL>	Conditionally either renders or skips the content of the tag, depending on the document context.
<IKB:MASTERSELECT>	Renders a button for selecting the parent document for the current document.
<IKB:CONDITION>	Renders the conditions defined for the search form field. Only applicable to Search sources.

<IKB:IMAGEVARIANT>	Renders image variants defined in a form. Only applicable to HTML-based forms for both modes (insert and update)
<ORACLE>	Executes a PL/SQL code fragment, and renders the output from the execution.

Generally, you can add any HTML-attribute to the iKnowBase tag, and it will be rendered as part of the output.

ikb:form

The <ikb:form> tag renders an HTML form-tag with the required attributes (action, method) set appropriately.

ikb:formname

The <ikb:formname> tag renders the form name, either as set in the setup of the RunForm portlet or using a default value of ï¿½form_<formId>".

ikb:label

The <ikb:label> tag will render the label of a form field, as specified in the forms configuration. The actual <ikb:label>-tag will be replaced with a tag, while the value ï¿½IKB:PROMPT" inside the tag will be replaced with the actual value.

Example:

```
<IKB:LABEL id="-344" class="label">IKB:PROMPT</IKB:LABEL>
```

Renders as:

```
<span class="label">TheSpecifiedLabel</span>
```

ikb:input

The <ikb:input> tag renders a html text field that can be used for editing a form field.

Example:

```
<IKB:INPUT id="-344" />
```

ikb:dimension

The <ikb:dimension> tag renders an iconic button that will open a dimension selector for selecting more values for a form field. Use this along with <ikb:value>, which will show the already selected values.

Example:

```
<IKB:DIMENSION id="800" imagesrc="/ressurs/list.gif"
 class="ikbFormDimension" />
<IKB:VALUE id="800" class="iKBFormInput" />
```

ikb:favorite_select

The <ikb:favorite_select> tag renders a drop down list of the last recently used values for the dimension attribute. It is typically used with <ikb:dimension>, which allow for the selection of values not in the drop down list.

Example:

```
<IKB:FAVORITE_SELECT id="802"  size="1" />
<IKB:DIMENSION id="802" imagesrc="/ressurs/list.gif" />
```

```
<IKB:VALUE id="800" class="iKBFormInput" />
```

ikb:space

The <ikb:space> tag renders a non-breaking space, the HTML entity

Example:

```
<IKB:SPACE>
```

ikb:calendar

The <ikb:calendar> tag renders a button used to open a calendar for selecting date values.

Example:

```
<IKB:CALENDAR id="764" imagesrc="/ressurs/evita/cal/calendar.gif"
 buttonclass="ikbFormCalendarButton" />
```

ikb:radio

The <ikb:radio> tag renders a set of radio buttons, corresponding to the options of a form field. The content of the tag will be repeated for each value, with the placeholders <IKB:INPUT_ID> and <IKB:INPUT_PROMPT> being replaced with the actual ID and prompt for each value.

Example:

```
<IKB:RADIO id="802">
  <IKB:INPUT_ID>
  <span class="label">IKB:INPUT_PROMPT</span>
</IKB:RADIO>
```

ikb:check

The <ikb:check> tag renders a set of checkboxes, corresponding to the options of a form field. The content of the tag will be repeated for each value, with the placeholders <IKB:INPUT_ID> and <IKB:INPUT_PROMPT> being replaced with the actual ID and prompt for each value.

Example:

```
<IKB:CHECK id="-142">
  <IKB:INPUT_ID class="ikbCheck" maxlength="20" size="20">
  <span class="label">IKB:INPUT_PROMPT</span>
</IKB:CHECK>
```

ikb:select

The <ikb:select> tag renders a select box for a form field. The select box may allow either single or multiple select, depending on the specified options.

Examples of a single select box:

```
<IKB:SELECT id="803"  size="1" />
```

Example of a multi-select box.

```
<IKB:SELECT id="-161" class="ikbSelect" size="4" multiple/>
```

ikb:button

The <ikb:button> tag renders a button for one of the default actions of the form. Specify which type of button it is by using the id-attribute of the <ikb:button>-tag.

Id	**Description**

SAVE	When the user clicks on this button the information given in the form will be saved, and the form will close. Applicable to Forms.
APPLY	When the user clicks on this button the information given in the form will be saved, and the form will remain open.Applicable to Forms.
NEXT	When the user clicks on this button the changes will be saved and he will navigate to the next Form in the Task wizard. Applicable to Forms which are part of a Task wizard.
PREVIOUS	When the user clicks on this button he the changes will be saved and he will navigate to the previous Form in the Task wizard. Applicable to Forms which are part of a Task wizard.
FINISH	When the user clicks on this button the changes will be saved, and the Task wizard will close. Applicable to Forms which are part of a Task wizard.
SEARCH	When the user clicks on this button the search will be performed with the search criteria given in the search form. Applicable to Search sources.
STORE	When the user clicks on this button the search criterias will be stored. They can be used to support saved search functionality. Applicable to Search sources.
RESET	When the user clicks on this button the changes performed in the form will be reset to the original values. Applicable to Forms and Search sources.

Example of a save button:

```
<IKB:BUTTON id="SAVE" class="ikbFormResetButton" value="Save" />
```

Example of a cancel button:

```
<IKB:BUTTON id="RESET" class="ikbFormResetButton" value="Cancel" />
```

Language support can be implemented by entering a NLS-string identification as the value of the tag:

```
<IKB:BUTTON id="RESET" value="#cs.reset#" />
```

ikb:related_object

The <ikb:related_object> tag renders an iconic button that will open a picklist for selecting related document objects for a form field. Use this along with <ikb:value>, which will show the already selected values.

Example:

```
<IKB:related_object id="800" imagesrc="/ressurs/list.gif"
 class="ikbFormDimension" />
<IKB:VALUE id="800" class="iKBFormInput" />
```

ikb:related_image

The <ikb:related_image> tag renders an iconic button that will open the image archive for selecting images for a form field. Use this along with <ikb:value>, which will show the already selected values.

Example:

```
<IKB:RELATED_IMAGE id="800" imagesrc="/ressurs/list.gif"
 class="ikbFormDimension" />
```

```
<IKB:VALUE id="800" class="iKBFormInput" />
```

ikb:file

The <ikb:file> tag renders a ï¿½browse" button for uploading file content into a form field. Use along with <ikb:filelink>, which shows the current filename (if any).

Example:

```
<IKB:FILE id="675"  />
```

ikb:filelink

The <ikb:filelink> tag renders the filename of the current file of the document being edited.

Example:

```
<IKB:FILELINK id="675" />
```

ikb:fileurl

The <ikb:fileurl> tag renders the url to the current file of the document being edited. Can be used within a File attribute or a document file attribute

Example:

```
<img border="2" src="<IKB:FILEURL id="675" />">
<a href="<IKB:FILEURL id="675" />" title="This is an image">My title</a>
```

ikb:attachment

The <ikb:attachment> tag renders a ï¿½browse" button for uploading attachments to the document being edited.

Use along with <ikb:filelink>, which shows the current filename (if any).

Example:

```
<IKB:ATTACHMENT id="774"  />
```

ikb:textarea

The <ikb:textarea> tag renders a HTML <textarea>..</textarea> for a form field.

Example:

```
<IKB:TEXTAREA id="646" ROWS="rows" COLS="cols">IKB:VALUE</IKB:TEXTAREA>
```

ikb:imagevariant

The <ikb:imagevariant> tag renders image variants defined in a form. To defined image variants in a form, you should add scaling attributes for image variants. For optional variants, you should add request attributes as well. The image variants are rendered as checkboxes and attribute labels, for mandatory variants the checkboxes are disabled and checked. In update mode, you can access the generated image variants from links on the attribute labels.

Note: Images will not be upscaled to create image variants. If a variant is larger than the image itself, the image variant will not be available for selection in the form unless you have configured the variant to use the original image in such circumstances.

ikb:addattribute

The <ikb:addattribute> tag renders a select list of possible other attributes for the form. Use this for dynamic forms, where you want the user to be able to add other attributes to the form, at will.

Example:

```
<IKB:ADDATTRIBUTE id="-5"/>
<IKB:DISPLAY_ADD_ATTR id="-5"/>
```

ikb:display_add_attr

The <ikb:display_add_attr> tag renders a list of attributes added to the form at run-time. Use this for dynamic forms, where you want the user to be able to add other attributes to the form, at will.

Example:

```
<IKB:DISPLAY_ADD_ATTR id="-5"/>
```

ikb:prompt

The <ikb:prompt> tag renders the prompt defined for the form field.

Example:

```
<IKB:LABEL id="794">IKB:PROMPT</IKB:LABEL>
```

ikb:display

The <ikb:display> tag renders the value of a form field as a readable text.

Example:

```
<IKB:DISPLAY id="18">IKB:VALUE</IKB:DISPLAY>
```

ikb:conditional

The <ikb:conditional> tag conditionally either renders or skips the content of the tag, depending on the document context.

Example for ï¿½select folder", which is displayed only when uploading a document from a desktop productivity package (Microsoft Office):

```
<IKB:CONDITIONAL id="752"  />
  <IKB:DIMENSION id="752" imagesrc="/ressurs/list.gif"
 class="ikbFormDimension" />
</IKB:CONDITIONAL id="752">
```

Example for ï¿½select template", which runs only when a document is **not** uploaded from a desktop productivity package:

```
<IKB:CONDITIONAL id="753" />
  <IKB:SELECT id="753"  size="1" />
</IKB:CONDITIONAL id="753">
```

ikb:masterselect

The <ikb:masterselect> renders a button for selecting the parent document for the current document.

ikb:condition

The <ikb:condition> tag renders the conditions defined for the search form field.

Example:

```
<IKB:CONDITION id="-344" class=""/>
```

This tag is only applicable to search sources.

oracle

The <oracle> tag executes any PL/SQL code fragment, and renders the output produced by the executed code. The standard HTP-package supplied with the database may be used.

All valid PL/SQL commands are permitted, and you can call other functions that are implemented in packets or self-contained functions.

Note: Stored functions are preferred over in-place code due to maintenance issues.

You can also send context-based parameters.

Variable	**Description**
:DOCUMENT_ID	If the document exists, the document ID is sent.
:PARENT_ID	DocumentID of the parent document, if the parent ID exists
:REFERENCE_PATH	Identifier for the portlet instance
:STYLE_ID	Identifier for the form definition
:TASK_ID	Identifier for the task wizard definition, when applicable
:STEP_ID	Identifier for the task step definition, when applicable
:LANGUAGE_ID	The current language code (from user, form, session, ...)
:USER_ID	Userid of the current user
:FORMNAME	The formname
:BACK_URL	The Back URL, as defined by Portal.

Example:

```
<ikb:form>
<table>
  <tr>
    <td>
      <p><ORACLE>
      Begin
          Htp.p('The time is now: '||to_char(sysdate,'hh24:mi'));
      End;
      </ORACLE></p>
    </td>
    <td>
      <p><ORACLE>
      Begin
          Htp.p( 'The number of sub-documents are: '
                   || subdocuments_count(:DOCUMENT_ID));
      End;
      </ORACLE></p>
    </td>
  </tr>
</table>
</ikb:form>
```

22. SOLR Schema model

Integration between SOLR and iKnowBase is implemented with usage of a solr configuration (in iKBStudio) where the mapping between iKnowBase attributes and the Solr schema is done.
To support the dynamic attribute model in iKnowBase, we use dynamic fields in Solr to store, index or make searchable the same attributes.
Also NLS-support is implemented with dynamic fields, and by default Norwegian and English is supported. If the iKnowBase instance supports more languages, we need to add the required fields for each language.

Overview of iKnowBase fields in solr/Schema.xml

Tag	type	Description	property
id	System	`Unique identifier for a solr document. Syntax is <application>-<documentId> where application is typically IKNOWBASE`	Stored, indexed
indextime	System	`Indicates when the document was last indexed in SOLR`	Stored, Indexed
document_id	Document	iKnowBase document id	Stored, indexed
url	Document	The URL to the document in iKnowBase	Stored
title	Document	The title in iKnowBase	Stored, indexed
description	Document	Description of the document	
body	Document	The body text or content from a file	
`type_<language>`	Document	Name of the information type	Stored,Indexed
`status_<language>`	Document	Document status	Stored,Indexed
acl_id	Document	ACL id on the document. -1 if null	Indexed
owner_id	Document	The ownerID of the document	Indexed
valid_from	Document	Indicated when the document is valid from	Stored, Indexed
`*_single_qc`	Dynamic	Used to build query completion fields. Single valued	Indexed
`*_multi_qc`	Dynamic	Used to build query completion fields. Multi valued	Indexed
`content1`	Query field	Contains the searchable title	Indexed
`content2`	Query field	Contains the searchable description	Indexed
`content3`	Query field	Contains the searchable body text	Indexed

content4 or content4_<language>	Query field	Contains the searchable attribute values	Indexed
stem1	Query field	Stemmer field for title	Indexed
stem2	Query field	Stemmer field for description	Indexed
stem3	Query field	Stemmer field for body	Indexed
stem4 or stem4_<language>	Query field	Stemmer field for attribute values	Indexed
*_search<_language>	Attribute	Makes the value searchable	
*_index<_language>	Attribute	Index the value. Can be used as a query condition	Indexed
*_store<_language>	Attribute	Stores the value. Used for presentation	Store
*_search_index<_lan	Attribute	Combined field	Stored, Indexed
*_search_store<_language>	Attribute	Combined field	Stored
*_search_index_stor	Attribute	Combined field	Stored, Indexed
*_index_store<_language>	Attribute	Combined field	Stored,Indexed
*_index_store_date	Attribute	Date type to store and index dates	Store, Indexed
*_store_date	Attribute	Date type to store dates	Indexed
*_index_date	Attribute	Date type to index dates	Indexed
*_ident	Attribute	Integer type to handle id’s e.g dimensionID	Stored, Indexed
*_guid	Attribute	String type to handle GUID values e.g dimensionGuid	Stored, Indexed
*__path	Attribute	String type to handle dimension paths	Stored

23. Instant Server

The iKnowBase Instant Server is the web endpoint for real time asynchronous messaging.

In addition to the JavaScript API, see *Javascript library* > *iKnowBase.Instant* in the *iKnowBase API Reference*, which supports both publish and consume, the following APIs are available for publishing messages.

HTTP API

The HTTP API supports publishing to topics and is an easy way of publishing messages for HTTP clients when there is no need for subscribing to and receiving messages.

The HTTP API v1 has the following endpoints:

Endpoint	Authentication	Authentication type	Return value	Comment
/ikb$service/v1/publish	Optional*	_ikbUserToken	NONE	*Topics with UserList support requires authentication using Secure Token Engine Authentication.
/private/ikb$service/v1/publish	Required	Container	NONE	This is a application server protected URL. Clients will be prompted for authentication according to authentication setup.
/ikb$service/v1/requestUserList	Required	_ikbUserToken	JSON:List of UserReference	Request a filtered userlist for the specified topic.
/private/ikb$service/v1/requestUserList	Required	Container	JSON:List of UserReference	Request a filtered userlist for the specified topic.
/ikb$service/v1/requestUserListUpdate	Required	_ikbUserToken	JSON:List of UserReference	Request a full userlist for the specified topic. Same as the javascript client subscription option requestUserList=true.
/private/ikb$service/v1/requestUserListUpd	Required	Container	JSON:List of UserReference	Request a full userlist for the specified topic. Same as the javascript client subscription option requestUserList=true.

NOTE: For the old method “sendMessage” is deprecated and replaced by “publish”.

/ikb$service/v1/sendMessage:

Parameter	Description	Type	Required

topic	Topic to publish message on. This is the full topicName (<topic base name>?<topic options>).	String	YES
message	Text coded data to publish on channel.	String	YES
_ikbUserToken	Scure token identifying the client user.	String	NO*

/private/ikb$service/v1/sendMessage:

Parameter	Description	Type	Required
topic	Topic to publish message on. This is the full topicName (<topic base name>?<topic options>).	String	YES
message	Text coded data to publish on channel.	String	YES

Sending messageFormat=IKB

An example “message” parameter using sendMessage with messageFormat=IKB:

```
{"toUserName":"","messageType":"SAMPLE_MESSAGE","data":"Hi, how are you?"}
```

fromUser will be added to the IKBResponseMessage based on authentication.

/ikb$service/v1/requestUserList:

Parameter	Description	Type	Required
topic	Topic requested for user list. This is the full topicName (<topic base name>?<topic options>).	String	YES
userNames	Array of usernames.	String[]	NO
_ikbUserToken	Scure token identifying the client user.	String	YES

Return value: List of UserReference as JSON.

/private/ikb$service/v1/requestUserList:

Parameter	Description	Type	Required
topic	Topic requested for user list. This is the full topicName (<topic base name>?<topic options>).	String	YES
userNames	Array of usernames.	String[]	YES

Return value: List of UserReference as JSON.

/ikb$service/v1/requestUserListUpdate:

Parameter	Description	Type	Required
topic	Topic requested for user list. This is the full	String	YES

	topicName (<topic base name>?<topic options>).		
_ikbUserToken	Scure token identifying the client user.	String	YES

Return value: List of UserReference as JSON.

/private/ikb$service/v1/requestUserListUpdate:

Parameter	Description	Type	Required
topic	Topic requested for user list. This is the full topicName (<topic base name>?<topic options>).	String	YES

Return value: List of UserReference as JSON.

PLSQL API

See also: *PL/SQL ContentServices API* and the package `IKB_INSTANT`.

Publish message

The PLSQL API supports publishing to topics using procedure IKB_INSTANT.PUBLISH with the following parameters.

Parameter	Description	Type	Required	Message formats
p_topic	Topic to publish message on. This is the full topicName (<topic base name>?<topic options>).	VARCHAR2	YES	TEXT,IKB
p_data	Text coded data to publish on channel.	CLOB	YES	TEXT,IKB
p_from_username	The userName that published the message.	VARCHAR2	NO*	IKB
p_to_username	The target userName when publishing a private message.	VARCHAR2	NO	IKB
p_message_type	A message type used by the application.	VARCHAR2	NO	IKB

Example procedure for integration with iKnowBase Event using TEXT message:

```
CREATE OR REPLACE
PROCEDURE publish_to_instant_topic1_text
(
  PARAMS      IN    OT_EVENTPARAMS
  , OLDREC    IN    OT_DOCUMENT
)
IS
  P_TOPIC         VARCHAR2(200);
  P_DATA          CLOB;
  P_FROM_USERNAME VARCHAR2(200);
```

```
  P_TO_USERNAME   VARCHAR2(200);
  P_MESSAGE_TYPE  VARCHAR2(200);
BEGIN
  P_TOPIC         := '/Topic1?messageFormat=TEXT';
  P_DATA          := 'Event triggered due to id: '||params.object_id;
  P_FROM_USERNAME := NULL;
  P_TO_USERNAME   := NULL;
  P_MESSAGE_TYPE  := NULL;
  IKB_INSTANT.PUBLISH(
      P_TOPIC
      , p_DATA
      , P_FROM_USERNAME
      , P_TO_USERNAME
      , P_MESSAGE_TYPE
  );
  COMMIT;
END;
```

Example procedure for integration with iKnowBase Event using IKB message:

```
CREATE OR REPLACE
PROCEDURE publish_to_instant_topic1_ikb
(
  PARAMS      IN    OT_EVENTPARAMS
  , OLDREC    IN    OT_DOCUMENT
)
IS
  P_TOPIC         VARCHAR2(200);
  P_DATA          CLOB;
  P_FROM_USERNAME VARCHAR2(200);
  P_TO_USERNAME   VARCHAR2(200);
  P_MESSAGE_TYPE  VARCHAR2(200);
BEGIN
  P_TOPIC         := '/Topic1?messageFormat=IKB';
  P_DATA          := 'Event triggered due to id: '||params.object_id;
  P_FROM_USERNAME := 'IKBUSER1';
  P_TO_USERNAME   := NULL;
  P_MESSAGE_TYPE  := 'SAMPLE_MESSAGE';
  IKB_INSTANT.PUBLISH(
      P_TOPIC
      , p_DATA
      , P_FROM_USERNAME
      , P_TO_USERNAME
      , P_MESSAGE_TYPE
  );
  COMMIT;
END;
```

NOTE: You may need to issue “set define off” due to ampersands (&) in P_TOPIC.

Server request: requestUserList

The PLSQL API supports the server request “requestUserList” for UserList topics.

Parameter	Description	Type	Required	Default value
p_topic	Topic requested for user list. This is the full topicName (<topic base name>?<topic options>).	VARCHAR2	YES	null

p_from_username	The userName that requests the list.	VARCHAR2	YES	null
p_user_names	Specify to only receive a filtered list of users.	IKB_PORTAL_API.VC_ARR	NO	empty array
p_timeout	Specify to only receive a filtered list of users.	IKB_PORTAL_API.VC_ARR	NO	10 (seconds)

The call will return CT_USEREFERENCE, which is a list of OT_USERREFERENCE of users currently registered as online on the topic.

Example statement for calling the requestUserList:

```
DECLARE
  P_TOPIC VARCHAR2(200);
  P_USER_NAMES IKB_PORTAL_API.VC_ARR;
  P_FROM_USERNAME VARCHAR2(200);
  P_TIMEOUT NUMBER;
  L_USER_REFERENCES CT_USERREFERENCE;
BEGIN
  P_TOPIC := '/Topic1?messageFormat=IKB&userList=true';
  P_USER_NAMES := ikb_portal_api.empty_vc_arr;
  P_FROM_USERNAME := 'ORCLADMIN';
  P_TIMEOUT := 10;
  L_USER_REFERENCES := IKB_INSTANT.REQUEST_USERLIST(
    P_TOPIC => P_TOPIC,
    P_USER_NAMES => P_USER_NAMES,
    P_FROM_USERNAME => P_FROM_USERNAME,
    P_TIMEOUT => P_TIMEOUT
  );
  DBMS_OUTPUT.PUT_LINE('Number of user references returned: ' ||
 l_user_references.COUNT);
  FOR i IN 1 .. L_USER_REFERENCES.COUNT
  LOOP
     DBMS_OUTPUT.PUT_LINE('id=' || L_USER_REFERENCES(i).id ||
 '; guid=' || L_USER_REFERENCES(i).guid || '; username=' ||
 L_USER_REFERENCES(i).username || '; dn=' || L_USER_REFERENCES(i).dn || ';
 label=' || L_USER_REFERENCES(i).label);
  END LOOP;
END;
```

NOTE: You may need to issue “set define off” due to ampersands (&) in P_TOPIC.

Server request: requestUserListUpdate

The PLSQL API supports the server request “requestUserListUpdate” for UserList topics. This call produces the exact same output as the javascript client subscription option requestUserList=true.

Parameter	Description	Type	Required	Default value
p_topic	Topic requested for user list. This is the full topicName (<topic base name>?<topic options>).	VARCHAR2	YES	null
p_from_username	The userName that requests the list.	VARCHAR2	YES	null
p_timeout	Specify to only receive a filtered list of users.	IKB_PORTAL_API.VC_ARR	NO	10 (seconds)

The call will return CT_USEREFERENCE, which is a list of OT_USERREFERENCE of users currently registered as online on the topic.

Example statement for calling the requestUserList:

```
DECLARE
  P_TOPIC VARCHAR2(200);
  P_FROM_USERNAME VARCHAR2(200);
  P_TIMEOUT NUMBER;
  L_USER_REFERENCES CT_USERREFERENCE;
BEGIN
  P_TOPIC := '/Topic1?messageFormat=IKB&userList=true';
  P_FROM_USERNAME := 'ORCLADMIN';
  P_TIMEOUT := 10;
  L_USER_REFERENCES := IKB_INSTANT.REQUEST_USERLIST_UPDATE(
    P_TOPIC => P_TOPIC,
    P_FROM_USERNAME => P_FROM_USERNAME,
    P_TIMEOUT => P_TIMEOUT
  );
  DBMS_OUTPUT.PUT_LINE('Number of user references returned: ' ||
 l_user_references.COUNT);
  FOR i IN 1 .. L_USER_REFERENCES.COUNT
  LOOP
     DBMS_OUTPUT.PUT_LINE('id=' || L_USER_REFERENCES(i).id ||
 '; guid=' || L_USER_REFERENCES(i).guid || '; username=' ||
 L_USER_REFERENCES(i).username || '; dn=' || L_USER_REFERENCES(i).dn || ';
 label=' || L_USER_REFERENCES(i).label);
  END LOOP;
END;
```

NOTE: You may need to issue “set define off” due to ampersands (&) in P_TOPIC.