

iKnowBase®

DevelopmentGuide



1. iKnowBase Development Guide

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

1. iKnowBase Development Guide.....	2
2. Introduction.....	7
1. Preface.....	7
1. About this guide.....	7
2. Audience.....	7
3. Conventions used in this guide.....	7
2. Overview.....	7
1. Development.....	7
2. Meta-modeling.....	7
3. iKnowBase Development Studio.....	9
1. The Work Area.....	9
1. Navigation pane.....	9
2. List pane.....	10
3. Edit pane.....	10
2. Navigating the Work Area.....	13
1. Development Tools and Advanced.....	13
2. User Directory and Metadata Management.....	13
3. Searching for Information.....	14
1. Development Tools and Advanced.....	14
2. User Directory and Metadata Management.....	15
4. Sorting the lists.....	15
5. Basic Tasks.....	15
1. Creating a New Object.....	15
2. Viewing an Object.....	16
3. Copying an Object.....	17
4. Editing an object.....	17
5. Deleting an object.....	18
6. Adding an object to a patch set.....	19
7. Displaying identifiers, audit- and storage information.....	19
4. iKnowBase Content Studio.....	21
1. Image Variants.....	21
1. Add Image Variant Definition.....	21
2. Change Image Variant Definition.....	21
3. Remove Image Variant Definition.....	22
4. Create Image Variants.....	22
2. Publish links.....	22
3. Configuration options.....	22
5. iKnowBase Content Services.....	23
6. Building iKnowBase Applications.....	25
1. Key components.....	25
2. Building Web Pages.....	25
1. Standalone Pages.....	25
2. Embedded pages.....	25
3. Layout Pages.....	26
3. Adding Content to Web Pages.....	26
4. Tooltip functions.....	26
1. Predefined tooltip functions.....	27
2. Define your own tooltip functions.....	27
5. Executing business logic.....	27
1. Script Targets.....	27
7. Using Scripts and Templates.....	28
1. Template concepts.....	28
2. Available components.....	28
3. Model objects.....	28
4. Available template languages.....	28
1. FreeMarker.....	29

2. Groovy.....	29
1. Implicit objects.....	29
2. Examples.....	29
3. JavaScript.....	31
4. HTML script.....	31
5. Template chaining.....	31
6. Error handling.....	31
8. Advanced Configuration.....	32
1. Domains.....	32
1. The Default Domain.....	32
2. Custom Access Control.....	32
3. Public user.....	32
4. Weight on free-text search.....	32
2. Attribute Security Function.....	33
1. Enable and Disable the Attribute Security Function.....	33
2. Signature of the Attribute Security Function.....	33
3. Run custom access control.....	33
1. Enable and Disable the custom access control.....	34
2. Signature of run_custom_access_control.....	34
4. Default common scripts configuration.....	34
1. jQuery.....	34
9. Using Apache Solr for Content Search.....	36
1. Concept.....	36
2. Installation and setup.....	37
1. iKnowBase security.....	37
3. Configuring the indexing process.....	37
4. Building a search page.....	37
1. Sample search page structure.....	38
5. Configuring search suggestions.....	38
6. Monitoring the Solr solution.....	39
10. Using iKnowBase Instant.....	40
1. Concept.....	40
1. JavaScript API and transport mechanisms.....	40
1. Synchronous HTTP.....	40
2. Asynchronous HTTP.....	40
1. long-polling.....	41
2. websocket.....	41
3. Transport failover.....	41
4. Blocking and non-blocking I/O.....	42
2. Topics.....	42
3. Message formats.....	42
4. Public and private messages.....	43
5. User list support.....	43
1. requestUserList.....	44
2. subscribeUserListChanges.....	44
6. APIs.....	45
2. Installation and setup.....	45
3. Deployment and scaling.....	45
4. Security.....	46
1. Authentication.....	46
1. Authenticating with Secure Token.....	46
2. Authenticating with direct application container authentication.....	46
2. Authorization.....	46
3. Private messages.....	47
5. Monitoring.....	47
6. Building solutions.....	47
1. Project wide settings.....	47
1. Project wide sample basic.....	47
2. Project wide sample with secure token authentication.....	47
3. Project wide sample with secure token authentication and CORS enabled.....	48
4. Instant subscribe with CORS.....	48

2. Simple Web to Web Client using text messages.....	49
3. Simple Web to Web Client using IKB messages and secure token authentication.....	50
4. Chat With User List Support.....	51
5. Content updates notifications Using Database Integration.....	54
1. Instant publish/subscribe sample code.....	54
2. Event.....	55
3. IKB database function mapping.....	55
4. Database procedure PUBLISH_TOPIC1.....	55
6. Livefeed Using Database Integration.....	55
7. Livefeed Using Cache Refresh Events.....	56
11. Performance.....	57
1. Concept.....	57
2. Monitoring.....	57
3. Caching on the client.....	57
1. Static shared resources (/ressurs).....	57
2. Caching content from the content server (/Content).....	58
1. HTTP Response cache directives to client based on client input.....	58
2. HTTP Response cache directives to client for autogenerated URLs to iKnowBase content.....	58
4. Caching on the application server.....	58
1. Metadata Cache.....	58
2. Page engine content cache.....	58
1. Configure content caching.....	58
2. Access control.....	59
3. Refreshing the content cache.....	59
4. Instant notifications of content cache updates.....	59
5. Setup.....	60
3. Clustering and the application server cache.....	60
5. Render Strategy.....	60
12. Integrating with an LDAP Directory.....	61
1. Understanding LDAP synchronization.....	61
2. Synchronizing users.....	61
1. User objects.....	61
2. Organization objects.....	62
3. Optional Attributes.....	62
4. Pre and Post Plug-Ins.....	62
1. Pre user synch plug-in.....	62
2. Post user synch plug-in.....	63
3. Synchronizing groups.....	64
1. Decide which groups to map.....	64
2. Hierarchical groups.....	64
3. Scheduling a Synchronization.....	64
4. LDAP provider specific requirements - OpenDJ.....	65
13. Integrating with Oracle Secure Enterprise Search.....	66
1. Concept.....	66
2. Install the iKnowBase adapter into Oracle SES.....	66
3. Create a source.....	67
4. Schedule a source for indexing.....	67
5. Security.....	67
6. Search User Interface.....	67
7. Custom filter plugin.....	68
14. Integrating with an E-mail Server.....	69
1. Configuring E-mail Accounts	69
2. Process Flow of the E-mail Reader.....	69
3. E-mail Reader data model.....	70
4. Message Handler.....	70
5. Reliable Messaging.....	71
6. Running the iKnowBase E-mail Reader.....	71
7. Configuring logging.....	72
15. Integration Using External Attribute Types.....	73
1. External Source API.....	73

1. Example: Topic Map API.....	74
16.iKnowBase Service API.....	77
1. Security considerations.....	77
2. API documentation.....	77
3. Sample code.....	77
1. Instantiate a reference object.....	77
1. Java.....	77
2. PL/SQL.....	77
2. Instantiate a document attribute.....	77
1. Java.....	77
2. PL/SQL.....	78
3. Document properties.....	78
1. Java.....	78
2. PL/SQL.....	78
4. Create document.....	79
1. Java.....	79
2. PL/SQL.....	79
5. Delete document attribute.....	80
1. Java.....	80
2. PL/SQL.....	80
17.iKnowBase Process Services for Activiti.....	81
1. Preparations.....	81
2. Developing process applications.....	81
1. Accessing Activiti data from a Template Viewer.....	82
2. Accessing iKnowBase data from a ScriptTask.....	82
3. Using a form to start processes and complete UserTasks.....	83
4. Associating an Activiti Process with an iKnowBase Document.....	84
18.iKnowBase Development Toolkit.....	85
1. Getting started.....	85
1. About the build system.....	85
2. Requirements.....	85
3. Distribution files for development.....	85
1. sample-build-root.....	85
2. sample-plugin-projects.....	85
4. Step-by-step setup.....	86
1. 1. Extract the provided development .zip files.....	86
2. 2. Configure build.....	86
1. gradle.properties.....	86
2. settings.gradle.....	86
3. build.gradle for your plugin projects.....	87
4. build.gradle for your root project.....	87
3. 3. Configure iKnowBase properties.....	87
4. 4. Test from command line.....	88
5. 5. Import in your favorite IDE.....	88
1. IntelliJ IDEA.....	88
2. Eclipse.....	88
6. 6. Setup iKnowBase web server run configuration in IDE.....	88
5. Assemble your plugins.....	88
6. Patch iKnowBase web application with your plugins.....	88
1. Run using the iKnowBase web server.....	89
7. Deploy.....	89
2. Plugin assembly.....	89
1. .jar plugin.....	89
2. .war plugin.....	90
3. Libraries.....	90
3. The iKnowBase application container.....	91
1. Extension points.....	91
2. Configuring the application container.....	91
3. Spring @Beans.....	92
1. Inject services.....	92
2. Inject configuration.....	92

4. Spring MVC Controller.....	92
1. Inject services and configurations.....	93
2. Inject method arguments.....	93
3. Security.....	93
5. Static web resources.....	94
6. Activiti.....	94
7. Using a plugin from iKnowBase Development Studio groovy components.....	94
8. Logging configuration.....	94
9. Solr Server.....	94
10. Form.....	95
1. Bean-to-document mapping.....	95
2. Form Processor.....	95
3. Form integration with iKnowBase Classic components.....	95
4. Available samples.....	96
1. iKnowBase Reference Plugins	96
5. Troubleshooting.....	96
1. Classpath issues in Eclipse based IDEs.....	96
2. Out of memory issues when building gradle project model.....	97
3. Accented (Norwegian) text loaded using resource bundle has wrong encoding.....	97
4. My Spring MVC Controller is not available as expected on the mapped path.....	97
5. Run from IDE does not start iKnowBase. Seems like application is missing.....	97
19. User Administration.....	98
1. Overview of User Administration.....	98
2. Manage Users.....	98
1. Automatic User Synchronization.....	98
2. Manual User Synchronization.....	98
3. Manage Groups.....	98
20. Security Administration.....	99
1. Overview of Security Administration.....	99
2. Manage Access Control Lists.....	99
3. Assign Access Control Lists to Objects.....	99
1. Overview of Document Access.....	100
2. Overview of Dimension Access.....	100
3. Overview of Form Access.....	100
4. Overview of Task Wizard Access.....	101
5. Overview of Page Access.....	101
6. Overview of Subsystem Access.....	101
21. Metadata Administration.....	102

2. Introduction

This chapter provides an overview about the information included in the Development Guide, the intended audience for this guide, and the conventions that are used in this guide.

Preface

About this guide

This guide describes how to develop, administer, and maintain iKnowBase solutions on top of an iKnowBase Content Services installation, as well as best practices for doing so.

- Chapter 2 provides an overview about the information included in the Development Guide, the intended audience for this guide, and the conventions that are used in this guide.
- Chapter 3 provides usage information on the development tools.
- Chapter 4-11 provides an overview of the iKnowBase Content Services. It describes how to build web pages, as well as advanced configuration, integration, and administration features.
- Chapter 12 provides information related to performance in an iKnowBase application.
- Chapter 13 provides an overview of the iKnowBase Process Services.
- Chapter 14-16 provides an overview of user, security, and metadata administration.

Audience

The information in this guide is intended for the iKnowBase developers, administrators, and meta-modelers.

Conventions used in this guide

- **Bold text** is used for names of user interface elements (such as names of fields, panes, windows, menus, buttons, dialog boxes) and what the user specifically selects, clicks, presses, or types.
- *Italic text* is used for titles of publications referenced in the text, emphasis (for example, a new term), and variables.
- **Note:** is used to indicate information that is important, but not hazard-related.

Overview

Development

As an iKnowBase developer you can build solutions using the iKnowBase Content Services and the iKnowBase Process Framework. A wide variety of solutions may be built, from simple web pages which present content to its users to advanced enterprise portals which provide a context-based, structured, and organized framework to integrate information, people, and processes within an organization.

The common tasks of the iKnowBase developer are related to mostly four areas:

- Building solutions, which involves configuration of components and building of pages with portlets
- Advanced configuration
- Integration with other systems
- Administration of the installation

Meta-modeling

Meta-modeling enables you to create a model or a structure for a portal. In iKnowBase, meta-modelers can also create users and user groups, and assign rights to them. These users can create, view, edit, or delete content in the portal. The meta-modelers are also referred to as administrators.

The tasks of the iKnowBase administrator are related mostly two areas:

- User and security administration, where you manage what users should have access to the system, and what permissions they should have.
- Metadata administration, where you manage metadata in the system.

Note: In order to execute the administration tasks, you must be given administrative rights to the system.

3. iKnowBase Development Studio

iKnowBase contains iKnowBase Development Studio, a suite of web-based tools for developers, administrators, and meta-modelers. This chapter describes the work area and the basic tasks you can perform using the tools.

Development Studio enables you to build and administer iKnowBase applications, and integrate them with external systems.

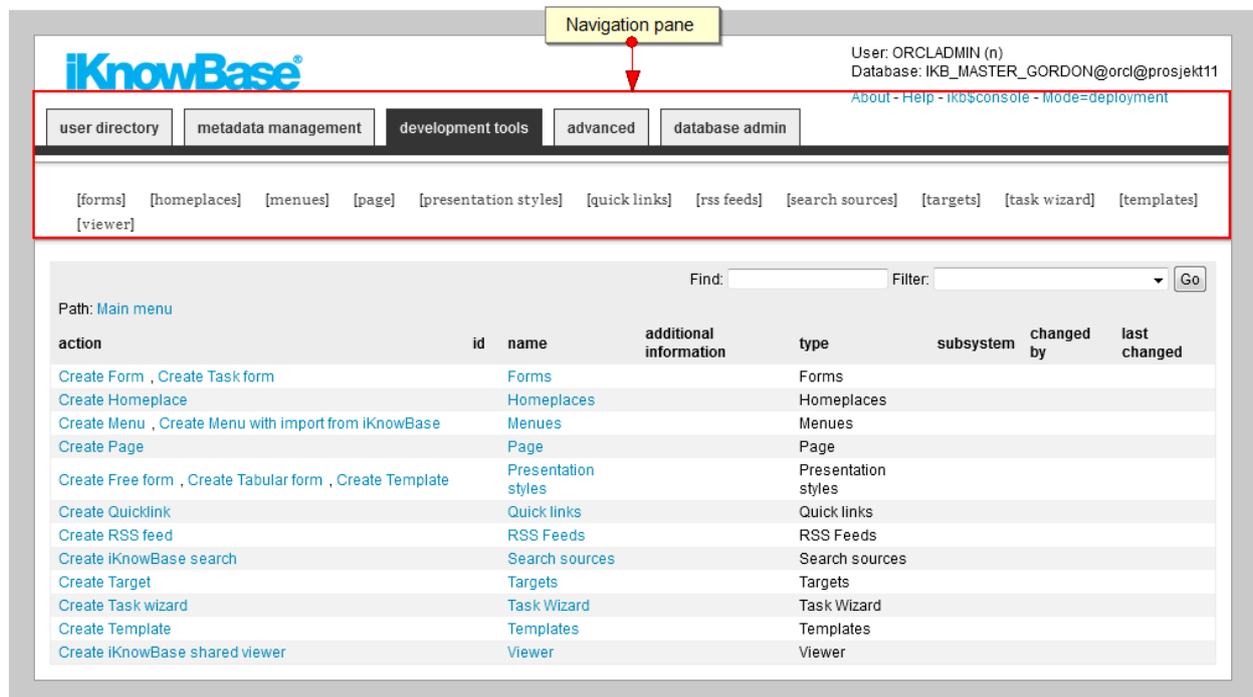
Development Studio will often be deployed as <http://www.example.com/ikbStudio>, but this may vary depending on installation choices.

When starting the Development Studio, you will be asked to log in. The exact mechanism for this will vary from installation to installation; please ask your support personnel for assistance.

The Work Area

The development work area consists of the **Navigation** pane, **List** pane, and **Edit** pane.

Navigation pane



The **Navigation** pane enables you to navigate between the different tools of the Development Studio, and between the different types of iKnowBase components. The **development tools** and **advanced** tabs represents the tools used for development of iKnowBase applications. The **user directory** and **metadata management** tabs represents the tools used for administration and meta-modelling.

Note: When you first enter the **development tools** and the **advanced** tabs you get access to the front page with the main menu and global search facilities. Click on a sub menu element in the **Navigation** pane to enter the work area for the corresponding component type. To return to the front page, click on the Main menu link in the **List** pane.

List pane

The screenshot shows the iKnowBase interface with the 'List pane' active. The navigation pane includes 'development tools' and 'advanced'. The main content area shows a table of RSS feeds. Annotations highlight various actions and filters.

action	id	name	additional information	type	subsystem	changed by	last changed
Edit , Copy , Delete	7667	Sample: RSS feed	http://www.lo.no	RSS Feed	iKB Demo application	ORCLADMIN	08.06.2009
Edit , Copy , Delete	9898		www.evita.no	RSS Feed	iKB Demo application	ORCLADMIN	08.06.2009

The **List** pane lists the existing components of the component type selected in the **Navigation** pane. If no component type is selected you are on the front page. Then the **List** pane contains either links to the different component types or a list of components matching the search filter, if given. The **List** pane enables you to filter the list and perform actions like create new components, and **edit**, **copy**, and **delete** existing components.

For the **development tools** and **advanced**, your location in the Development Studio is indicated by **Path**, which also includes a link to the front page.

Edit pane

The screenshot shows the iKnowBase interface with the 'Edit pane' active. The navigation pane includes 'development tools' and 'advanced'. The main content area shows a form for editing a menu. Annotations highlight various form fields and buttons.

menus: cs main menu

create and edit menus

Edit the menu by entering a new name or a description. Javascript is used to add your own scripts and is optional. The icon fields give you an option to add icons to menus and menu elements. You can in the presentation of the menu decide if you want to show them or not.

subsystem: iKB Content Studio

* name: CS Main Menu

* description: Main menu for iKnowBase Content Studio

javascript: [text area]

icon for menus: [width] [height]

icon for element: [width] [height]

icon for active element: [width] [height]

database information:

guid	717B21BAEDDF60CFE04000A17000798
id	42743
created	19.08.2009 11:08 by ORCLADMIN
updated	05.12.2011 12:26 by ORCLADMIN
master table	MENU_COMPONENT (Menus)
detail tables	MENU_ITEM

patch sets:

action	id	name	last built
Add to patch set			

You get access to the **Edit** pane by clicking on the **create** action or the **edit** action for a particular component in the **List** pane. The **Edit** pane enables you to create new components, or view details for and update existing components.

For existing components under **development tools** and **advanced**, the **Edit** pane is organized in tabs. Which tabs that are available, are dependent on the component type. The **edit** and **usage** tabs are available for all components. The **edit** tab contains the properties available at creation time. The **usage** tab shows a list of all objects that are dependent of the component being edited, as well as actions that will be taken if the component is deleted. The tabs are not available while creating a new component, but become available after initial save of the component. **Note:**Remember to save you work before you navigate to another tab within the **Edit** pane.

Mandatory fields are indicated with red stars.

Common action buttons available on the **Edit** pane are **Save (accesskey S)**, **Apply (accesskey S)**, **OK (accesskey X)**, **Copy**, **Delete** and **Cancel**. You can click on **Apply**, **Save**, or **OK** to save your work. If you use **Apply** or **Save** the **Edit** pane will remain open, and you can continue editing the object. If you use **OK** the **Edit** pane will close, and you return to the **List** pane for the object type.

Note: The work area for dimensions under **metadata management** differs from the other work areas. All three panes are present in the same work area. The **List** pane contains a tree structure rather than a list. You get access to the **Edit** pane by clicking on a dimension in the tree structure. The actions for **Add child dimension** and **Delete dimension** are available in the **Edit** for a particular dimension.

iKnowBase User: ORCLADMIN (us)
Database: IKB_MASTER_GORDON@ordl@projekt11
[About](#) - [Help](#) - [IKBconsole](#) - [Mode=deployment](#)

user directory | **metadata management** | development tools | advanced | database admin | help

information types | attributes | **dimensions** | dimension types | valuelists | site

Navigation pane

Expand/Collapse

- [-] iKnowBase
 - [-] Organization
 - [-] Customers
 - [-] **Products**
 - [-] Quality System
 - [-] Subjects
 - [-] Resources
 - [-] Favorites

List pane

Edit pane

products Action buttons for this dimension → Save (S) Save and close (X) Cancel (ESC)

actions

Add child dimension Delete dimension ← Action buttons for this dimension

label and description

prompt (us) Products

prompt (kh) Products

prompt (n) Produkter
Struktur for virksomhetens produkter, varer og tjenester.

identity

subsystem

external key

structure

parent //iKnowBase [Change](#)

security

access control list [Change](#) [Clear](#)

dimension information

sort key 20

dimension types

Actions: [Add dimension type](#) [Add multiple dimensions types](#)

actions	id	label	external key
remove	8288	IKB Information area	

database information

guid A5DF43A0CCD141368AE7524C31F3E6B7

id 2157

created 07.06.2005 10:05:25 by ORCLADMIN

updated 01.12.2011 13:47:58 by ORCLADMIN

master table DIMENSIONS

detail tables DIMENSION_DIMENSION_TYPE

Navigating the Work Area

The screenshot shows the iKnowBase interface. At the top right, it displays 'User: ORCLADMIN (n)' and 'Database: IKB_MASTER_GORDON@ordl@projekt11'. Below this is a navigation pane with tabs for 'user directory', 'metadata management', 'development tools', 'advanced', and 'database admin'. The 'development tools' tab is currently selected. Below the tabs is a sub-menu with links for '[forms]', '[homeplaces]', '[menus]', '[page]', '[presentation styles]', '[quick links]', '[rss feeds]', '[search sources]', '[targets]', '[task wizard]', '[templates]', and '[viewer]'. The main content area shows a table of actions with columns: action, id, name, additional information, type, subsystem, changed by, and last changed.

action	id	name	additional information	type	subsystem	changed by	last changed
Create Form , Create Task form		Forms		Forms			
Create Homeplace		Homeplaces		Homeplaces			
Create Menu , Create Menu with import from iKnowBase		Menus		Menus			
Create Page		Page		Page			
Create Free form , Create Tabular form , Create Template		Presentation styles		Presentation styles			
Create Quicklink		Quick links		Quick links			
Create RSS feed		RSS Feeds		RSS Feeds			
Create iKnowBase search		Search sources		Search sources			
Create Target		Targets		Targets			
Create Task wizard		Task Wizard		Task Wizard			
Create Template		Templates		Templates			
Create iKnowBase shared viewer		Viewer		Viewer			

In the top right corner you will see the username you are logged in as, as well as information about the database you are connected to.

The tabs of the top menu correspond to the four major tools of the iKnowBase Development Studio: **user directory**, **development tools**, **advanced** and **metadata management**. To navigate to the subpage for a particular tool, click on the tab representing that tool. The current tab selection is highlighted with a dark background.

Underneath the tabs is another set of navigation links; the sub menu for the current tool. These menu elements correspond to the various metadata or object types that you can maintain using the tool. To navigate to the **List** pane for a particular metadata or object type, click on the menu element representing that type. The current selection is highlighted with a triangle underneath.

Development Tools and Advanced

The **Edit** pane is organized in tabs. The current tab selection is highlighted with a dark background. To navigate to the correct tab of the **Edit** pane, click on the tab.

Note:When you navigate between the tabs of the **Edit** pane, unsaved work is lost.

In **Edit** panes for components referring to other components, it is possible to navigate directly to the **Edit** pane of the referenced component.

Behind a property referencing another component there is an arrow icon



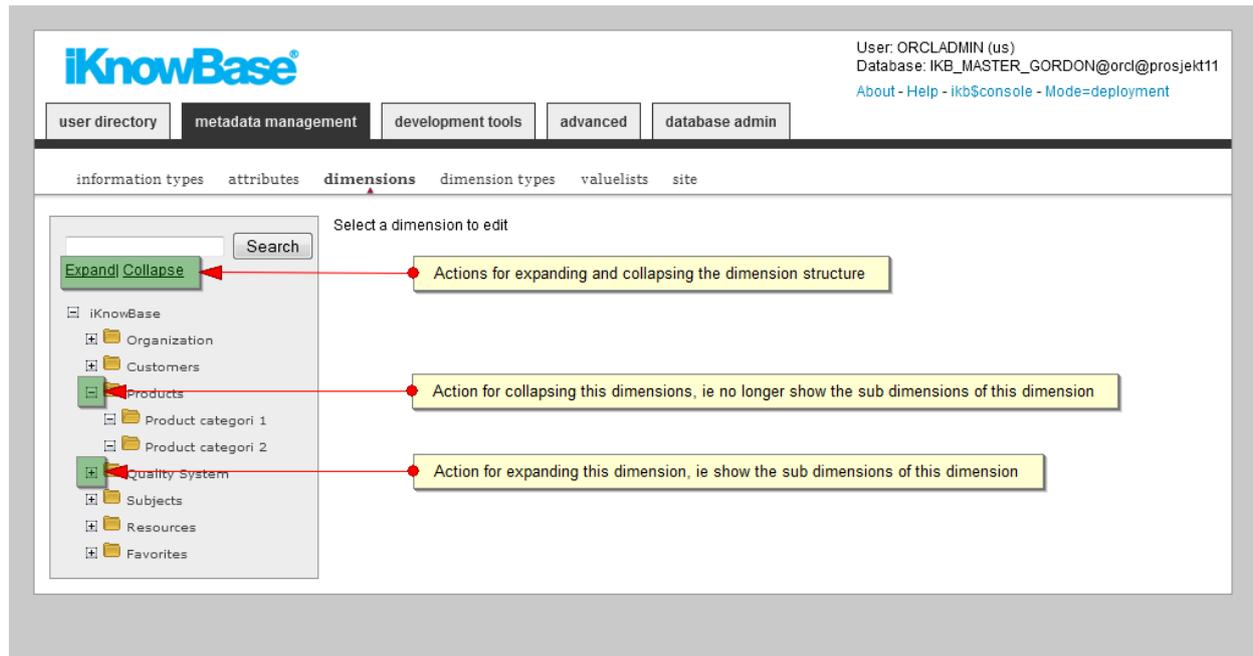
Click on this icon to navigate to the referenced component. This will cause a new browser window to open.

User Directory and Metadata Management

In the **List** pane, list navigation links are present in the top right corner. To navigate to a given page, click on a number, which represents the page. To navigate to the first or last page of the list, click on the **<<** or **>>** respectively. To browse the list page by page forward or backward, click on the **>** or **<** respectively.

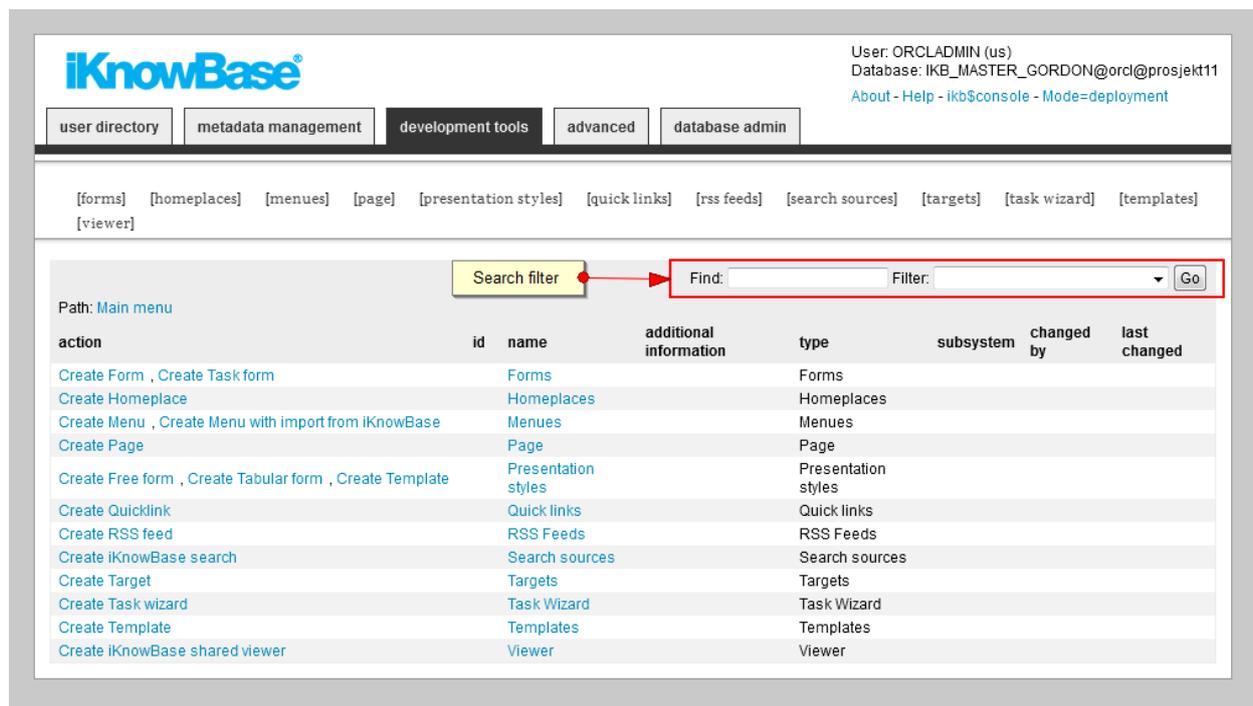
In the **List** pane for dimensions / dimension navigators, click on the **Expand** action to show all the dimensions in the structure, click on the **Collapse** action to collapse the structure and show only the top dimension. Click on the **+** to expand a given dimension. This will show the sub dimensions of the dimension. Click on the **-** to collapse a given dimension. This will hide the sub dimensions of the

dimension. If you want to filter the dimension structure, type the appropriate filter that you want to apply in the search filter, and press **ENTER** or click the **Search** button.



Searching for Information

Development Tools and Advanced



On the front page you will find a global search facility which enables you to search for components across all component types. If you know which type of component you are looking for, you can use the local search facility. This is available in the **List** pane for the given component type. Both free text search and a subsystem filter are available for both global and local searches. **Note:** The given subsystem filter will be applied until you select otherwise.

To search for information, perform the following steps:

1. On the front page or on the **List** pane for the type of component you are searching for, give the appropriate filter that you want to apply in the search filter.
2. Click **Go** or press **ENTER**. The list is filtered according to the applied search filter.

Note: If you do not specify any filter, all existing objects are displayed.

User Directory and Metadata Management

The screenshot shows the iKnowBase metadata management interface. The user is ORCLADMIN (us) and the database is IKB_MASTER_GORDON@orcl@prosjekt11. The interface includes navigation tabs for 'user directory', 'metadata management', 'development tools', 'advanced', and 'database admin'. Under 'metadata management', there are sub-tabs for 'information types', 'attributes', 'dimensions', 'dimension type', and 'site'. A 'Search filter' box is highlighted with a red box and an arrow pointing to it. Below the search filter is a table of dimension types with columns for 'actions', 'id', 'label', 'external key', 'changed by', and 'changed date'.

actions	id	label	external key	changed by	changed date
edit copy delete	8425	IKB Customer		ORCLADMIN	05.12.2011 10:24:25
edit copy delete	7659	IKB Departement	DIMTYPE_IKB_DEPARTMENT	ORCLADMIN	05.12.2011 10:24:25
edit copy delete	7661	IKB Group			01.07.2009 15:49:31
edit copy delete	8288	IKB Information area			05.12.2011 10:24:25
edit copy delete	8595	IKB Main process	IKB_PROCESS	ORCLADMIN	30.01.2009 08:51:29
edit copy delete	7660	IKB Person	DIMTYPE_IKB_PERSON	ORCLADMIN	05.12.2011 10:24:25
edit copy delete	15585	IKB Process Area	IKB_PROCESS_AREA	ORCLADMIN	05.12.2011 10:24:25
edit copy delete	8157	IKB Process Step	IKB_PROCESS_STEP	ORCLADMIN	30.01.2009 08:55:54
edit copy delete	15925	IKB Processview	IKB_PROCESSVIEW	ORCLADMIN	05.12.2011 10:24:25
edit copy delete	15586	IKB Sub Process	IKB_SUBPROCESS	ORCLADMIN	06.01.2010 12:35:40
edit copy delete	52010	IKB WebDav root folder	IKB_WEBDAV_FOLDER	ORCLADMIN	05.12.2011 10:24:25

To search for information, perform the following steps:

1. On the **Navigation** pane, click the type of metadata or user object in which you want to search for information.
2. In the **List** pane, type the appropriate filter that you want to apply in the search filter. The filter works on both label, id, external key, and guid for all objects, except dimensions. Dimensions can only be filtered by label.
3. Press **ENTER**. The list is filtered according to the applied search filter. If the criterion that you specify does not match any existing objects in the iKnowBase portal, the **No Records Found** message appears on the **List** pane.

If you do not specify any information in the search filter field, iKnowBase displays all existing objects.

Sorting the lists

You can sort the list in the **List** pane by clicking on the heading of the column you want to order by. The first click will result in ascending order. The next click will result in descending order. Sorted columns are indicated with up or down arrows next to the sort column.

Basic Tasks

This chapter describes the basic tasks you can perform using Development Studio.

Creating a New Object

The screenshot shows the iKnowBase development studio interface. The user is ORCLADMIN (us) and the database is IKB_MASTER_GORDON@orcl@prosjekt11. The interface includes navigation tabs for 'user directory', 'metadata management', 'development tools', 'advanced', and 'database admin'. Under 'development tools', there are sub-tabs for '[forms]', '[homeplaces]', '[menus]', '[page]', '[presentation styles]', '[quick links]', '[rss feeds]', '[search sources]', '[targets]', '[task wizard]', '[templates]', and '[viewer]'. A 'Create new...' button is highlighted with a red box and an arrow pointing to it. Below the button is a table of RSS feeds with columns for 'action', 'id', 'name', 'additional information', 'type', 'subsystem', 'changed by', and 'last changed'.

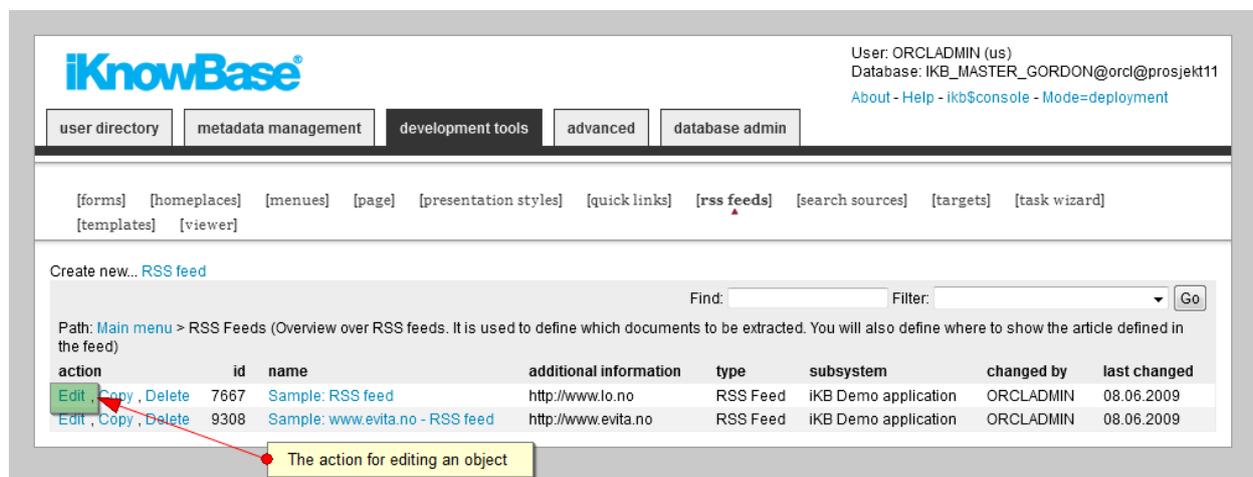
action	id	name	additional information	type	subsystem	changed by	last changed
Edit , Copy , Delete	7667	Sample: RSS feed	http://www.lo.no	RSS Feed	IKB Demo application	ORCLADMIN	08.06.2009
Edit , Copy , Delete	9308	Sample: www.evita.no - RSS feed	http://www.evita.no	RSS Feed	IKB Demo application	ORCLADMIN	08.06.2009

To create a new object, perform the following steps:

1. Navigate to the appropriate tool.
2. Click on the menu element for the appropriate object type. The **List** pane for the selected object type is opened.
3. Click on the  or **Create *objectType*** action, where *objectType* is the type of object you want to create. An empty **Edit** pane opens.
4. On the **Edit** pane, enter the appropriate information in each field. See the corresponding chapters in the reference documentation for information about the object type properties.
5. Click **Apply** or **Save**. iKnowBase creates the object and saves all the information that you provided for this object. The **Edit** pane remains open. In **development tools** it is now split into tabs with all the properties of the object available.

objectType

Viewing an Object



User: ORCLADMIN (us)
Database: IKB_MASTER_GORDON@orcl@prosjekt11
[About](#) - [Help](#) - [ikb\\$console](#) - [Mode=deployment](#)

user directory | metadata management | **development tools** | advanced | database admin

[forms] [homeplaces] [menus] [page] [presentation styles] [quick links] **[rss feeds]** [search sources] [targets] [task wizard]
[templates] [viewer]

Create new... [RSS feed](#)

Find: Filter: Go

Path: [Main menu](#) > [RSS Feeds](#) (Overview over RSS feeds. It is used to define which documents to be extracted. You will also define where to show the article defined in the feed)

action	id	name	additional information	type	subsystem	changed by	last changed
Edit , Copy , Delete	7667	Sample: RSS feed	http://www.lo.no	RSS Feed	iKB Demo application	ORCLADMIN	08.06.2009
Edit , Copy , Delete	9308	Sample: www.evita.no - RSS feed	http://www.evita.no	RSS Feed	iKB Demo application	ORCLADMIN	08.06.2009

The action for editing an object

You can view the properties of an existing object in iKnowBase.

To view the properties of an object, perform the following steps:

1. Navigate to the appropriate tool.
2. Click on the menu element for the appropriate object type. The **List** pane for the selected object type opens.
3. Click on the **edit** action for the appropriate object. The properties for the object appear in the **Edit** pane. See the corresponding chapters in the reference documentation for information about the object type properties.
4. Click **Cancel** when you are finished viewing the object. The **Edit** pane closes and the **List** pane appears.

Copying an Object

The screenshot shows the iKnowBase interface. At the top right, it displays 'User: ORCLADMIN (us)' and 'Database: IKB_MASTER_GORDON@orcl@prosjekt11'. Below this are navigation tabs: 'user directory', 'metadata management', 'development tools', 'advanced', and 'database admin'. A secondary navigation bar contains links like '[forms]', '[homeplaces]', '[menus]', '[page]', '[presentation styles]', '[quick links]', '[rss feeds]', '[search sources]', '[targets]', and '[task wizard]'. The main content area is titled 'Create new... RSS feed' and includes a search bar with 'Find:' and 'Filter:' fields. Below the search bar, the path is shown as 'Main menu > RSS Feeds'. A table lists RSS feeds with columns for 'action', 'id', 'name', 'additional information', 'type', 'subsystem', 'changed by', and 'last changed'. Two rows are visible: one for 'Sample: RSS feed' (id 7667) and one for 'Sample: www.evita.no - RSS feed' (id 9308). A red arrow points from a callout box labeled 'The action for copying an object' to the 'Copy' link in the 'action' column of the first row.

action	id	name	additional information	type	subsystem	changed by	last changed
Edit , Copy , Delete	7667	Sample: RSS feed	http://www.lo.no	RSS Feed	IKB Demo application	ORCLADMIN	08.06.2009
Edit , Copy , Delete	9308	Sample: www.evita.no - RSS feed	http://www.evita.no	RSS Feed	IKB Demo application	ORCLADMIN	08.06.2009

Instead of going through all the steps that you need to perform to create an object, you can create a copy of an object and then update the properties for that object.

To create a copy of an object, perform the following steps:

1. Navigate to the appropriate tool.
2. Click on the menu element for the appropriate object type. The **List** pane for the selected object type opens.
3. Click on the **copy** action for the object that you want to copy. iKnowBase creates a copy of the object with the name **ObjectName - copy**, where **ObjectName** is the name of the original object. The object appears in the list of existing iKnowBase objects.

Editing an object

This screenshot is identical to the one above, showing the iKnowBase interface with the 'RSS feeds' list. A red arrow points from a callout box labeled 'The action for editing an object' to the 'Edit' link in the 'action' column of the first row.

action	id	name	additional information	type	subsystem	changed by	last changed
Edit , Copy , Delete	7667	Sample: RSS feed	http://www.lo.no	RSS Feed	IKB Demo application	ORCLADMIN	08.06.2009
Edit , Copy , Delete	9308	Sample: www.evita.no - RSS feed	http://www.evita.no	RSS Feed	IKB Demo application	ORCLADMIN	08.06.2009

You can update the information that you provided for an object.

To update an object, perform the following steps:

1. Navigate to the appropriate tool.
2. Click on the menu element for the appropriate object type. The **List** pane for the selected object type opens.
3. Click on the **edit** action for the object that you want to update. The properties for the object appear in the **Edit** pane. See the corresponding chapters in the reference documentation for information about the object type properties.
4. On the **Edit** pane, edit the information that you want to update.
5. Click **Save**. iKnowBase updates the properties for this object.

Note: Click **Cancel** if you want to cancel the changes you've made.

Deleting an object

The screenshot shows the iKnowBase interface. At the top right, it displays 'User: ORCLADMIN (us)' and 'Database: IKB_MASTER_GORDON@ordl@prosjekt11'. Below this are navigation links: 'About - Help - ikb\$console - Mode=deployment'. The main navigation bar includes 'user directory', 'metadata management', 'development tools', 'advanced', and 'database admin'. A secondary navigation bar contains links for '[forms]', '[homeplaces]', '[menus]', '[page]', '[presentation styles]', '[quick links]', '[rss feeds]', '[search sources]', '[targets]', and '[task wizard]'. Below this, there are links for '[templates]' and '[viewer]'. The main content area is titled 'Create new... RSS feed'. It features a search bar with 'Find:' and 'Filter:' fields, and a 'Go' button. Below the search bar, the path is shown as 'Main menu > RSS Feeds (Overview over RSS feeds. It is used to define which documents to be extracted. You will also define where to show the article defined in the feed)'. A table lists RSS feeds with columns for 'action', 'id', 'name', 'additional information', 'type', 'subsystem', 'changed by', and 'last changed'. Two rows are visible: one with id 7667 and name 'Sample: RSS feed', and another with id 9308 and name 'Sample: www.evita.no - RSS feed'. The 'Delete' button in the 'action' column of the first row is highlighted with a green box. A red arrow points from a yellow callout box below the table to this 'Delete' button. The callout box contains the text 'The action for deleting an object'.

You can delete an object from iKnowBase.

To delete an object, perform the following steps:

1. Navigate to the appropriate tool.
2. Click on the menu element for the appropriate object type. The **List** pane for the selected object type opens.
3. Click on the **delete** action for the object that you want to delete. A confirmation page appears. On this page you will get a list of objects which reference the object to be deleted, together with information about which actions will be taken if the object is deleted.
4. Click **delete** to delete object. iKnowBase deletes the object.

Note: You can click **Cancel** to cancel the deletion.

The screenshot shows a confirmation dialog in iKnowBase. The title is 'delete "cs document archive - list" of type "presentation styles"?'. On the right side, there are 'Action buttons' with 'Delete' and 'Cancel' buttons. Below the title, there is a warning: 'delete "cs document archive - list" of type "presentation styles" Please note the following: The object might be in use even if its not listed here (e.g. custom code, hardcoding). Please be sure of this before deleting the object'. A table lists objects that reference the object to be deleted. The table has columns for 'description', 'type', 'reference', and 'if deleted?'. One row is visible with description 'Viewer/search definition', type 'iKnowBase shared viewer', reference 'CS Documents - Search dimension and under', and 'if deleted?' 'The referenced column will be set to null'. A red arrow points from a yellow callout box below the table to this row. The callout box contains the text 'List of objects referencing the object to be deleted, with information about which actions will be taken on delete'.

Adding an object to a patch set

The screenshot shows the iKnowBase 'edit' page for a dynamic parameter named 'ikb attrib: changed by'. The 'definition of dynamic parameters' section includes fields for subsystem (IKB Desktop), parameter type (iKnowBase parameter), name (IKB Attrib: Changed by), and description (IKB Attrib: Changed by). The 'database information' section shows the parameter's GUID, ID, creation date, and master table. A 'patch sets' table is visible at the bottom right, with a red arrow pointing to it from a 'Patch sets' label. The table contains one entry: action 'Add to patch set', id '60249', name 'Patch Jira-2043', and last built timestamp '18.02.2013 08:25:11'.

You can add an object to a patch set within each component type. The patch set can later be exported and installed somewhere else.

To add an object to a patch set, perform the following steps:

1. Navigate to the appropriate tool.
2. Click on the menu element for the appropriate object type. The **List** pane for the selected object type opens.
3. Click on the **edit** action for the object that you want to add to a patch set. Select the link **Add to Patchset** to select a patch set.
4. Click **Save** to add the object to the patch set(s). **Last build** displays the timestamp the patch set last was generated.
5. To delete an object from a patch set, press the delete icon, then press **Save**.

Displaying identifiers, audit- and storage information

The screenshot shows the iKnowBase 'edit' page for a homeplace named 'Arbeidsrom'. The 'description and general properties' section includes fields for subsystem (xNet Solution), sort key (10), name (Arbeidsrom), information type (Arbeidsrom), and homeplace rule (Use the homeplace for the current document). The 'define conditions' section shows a table with columns for actions, attribute, disabled?, sql-clause, condition, and value. The 'define where to find the values for parameters in then target' section shows a table with columns for type, name, and value. A red arrow points to the 'Identifiers, audit information and storage info' type. The 'database information' section shows the homeplace's GUID, ID, creation date, and master table. A 'patch sets' table is visible at the bottom right, with a red arrow pointing to it from a 'Patch sets' label. The table contains one entry: action 'delete', id '60249', name 'Patch Jira-2043', and last built timestamp '18.02.2013 08:25:11'.

All objects has a section where audit information, internal identifiers and storage information are displayed.

1. Navigate to the appropriate tool.
2. Click on the menu element for the appropriate object type. The **List** pane for the selected object type opens.

3. Click on the **edit** action for the object that you want to add to a patch set.
4. At the bottom of the first tab on the left side, database information are displayed.

Property	Description
Guid	Displays a globally unique identifier number for this component. This property appears after you save a component. You cannot update the information of this property.
Id	Displays a unique identifier number for this component. This property appears after you save a component. You cannot update the information of this property.
Created	Displays when the component was created and by whom it was created.
Updated	Displays when the component was last updated and by whom it was updated.
Master table	Main database table where the instance is stored
Detail tables	Displays other database tables where details for the object are stored.

4. iKnowBase Content Studio

iKnowBase contains iKnowBase Content Studio, a suite of web-based tools for publishers. For end user documentation of the iKnowBase Content Studio, see *iKnowBase Publishing Guide*.

Image Variants

An image variant is a scaling variant of an image. Image variants are created when inserting and updating image documents in iKnowBase, and are used when publishing content. You can configure an image variant to be required or optional. A required image variant will be created for an image upon each save of the image. An optional image variant will be created if requested by the end user.

The event “Image Scaling” triggers creation of image variants.

You may add new, change and remove existing image variant definitions. Image variant definitions are maintained in the iKnowBase Development Studio (development tools). For reference documentation see the Image Variant chapter of *iKnowBase Development Reference*.

Note: There must exist one required image variant with the property “Use this variant as thumbnail?” checked.

The image archive, which is part of the iKnowBase Content Studio, is by default configured with a set of image variant definitions.

This chapter describes the process of maintaining image variant definitions.

Add Image Variant Definition

To create a new image variant definition, perform the following steps:

1. Create an attribute with data type = FILE, and specify an external key. This attribute will be used to store the actual image variant for an image. It is referred to as the **storage attribute**. **Note:** You must define an external key for this attribute.
2. If the image variant is optional, create an attribute with data type = VALUELIST and value list = iKnowBase YesNo. This attribute will be used in the image form and allows the end user to decide if the image variant should be created. It is referred to as the **request attribute**. **Note:** You must use the iKnowBase YesNo valuelist for the **request attribute** in order to make it work.
3. Finally, create the new image variant definition. Choose the newly created **storage attribute**, and the **request attribute** if the variant is optional. Specify the scaling properties.

If you use iKnowBase Classic version of the image form (“CS Image”), you must add the image variant attribute(s):

1. Add the **storage attribute** to the form. Change the attribute rule so that the attribute is not insertable, updateable, or savable.
2. If the image variant is optional, add the **request attribute** to the form.

Now the new image variant definition is ready for use!

Note: If an image variant’s only purpose is to define a crop area to be used in image edit, no attributes are required.

Change Image Variant Definition

You can change the scaling properties of an image variant definition. This will not automatically affect existing images/image variants. But, will affect all images created or modified after the image variant definition is changed. To recreate image variants for existing images, see the chapter *Create Image Variants*.

You can also make a required image variant optional, and vice versa, by adding or removing the **request attribute** from the image variant definition. **Note:** You must update the image form accordingly. See the chapter *Add Image Variant* for further information.

Remove Image Variant Definition

You can remove an image variant definition. **Note:** You must update the image form accordingly. See the chapter *Add Image Variant* for further information.

After removal of an image variant, the corresponding attributes will behave as regular attributes and already created image variants will remain.

Create Image Variants

The event “Image Scaling” triggers creation of image variants. By default this event is configured to trigger on insert and update of an image document. In special circumstances, like after an upgrade or after a change in the image variant definitions, you can manually recreate all image variants in the system. This functionality is available in the edit pane for the event “Image Scaling” in the iKnowBase Development Studio. Click the **Run batch** button in on the **run batch** tab of the event “Image Scaling”. See the Event chapter of the *iKnowBase Development Reference* for further information.

Publish links

The quicklink used in Content Studio is configured as an iKnowBase Classic Quicklink “CS Publish links”. You should change this configuration to match your needs.

Note: Changes to this component must be reapplied after an iKnowBase upgrade.

Configuration options

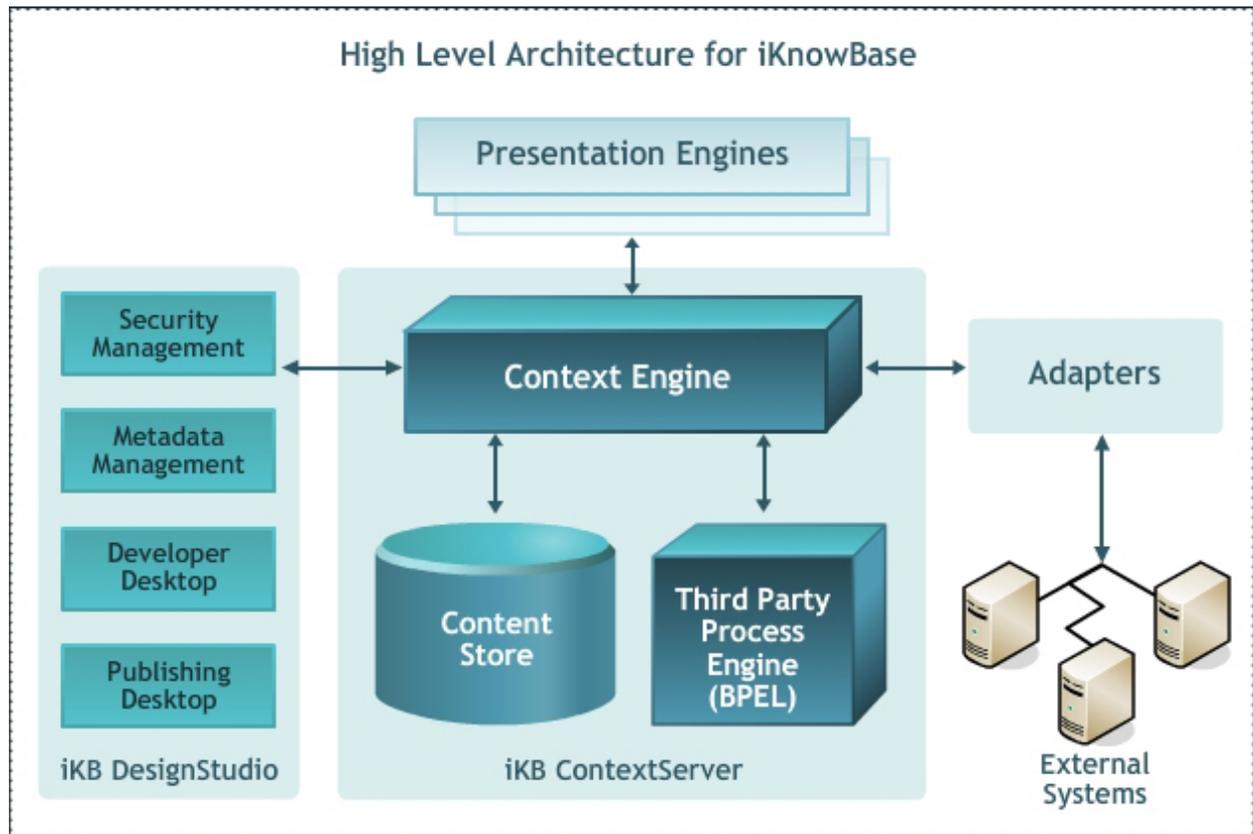
See *iKnowBase Installation Guide* for an overview over available configuration options for Content Studio.

5. iKnowBase Content Services

iKnowBase Content Services consist of a content store, a context engine, and presentation engines, as well as mechanisms for integration with external systems. It comes with administration tools for:

- Development
- Metadata
- User information
- Content, including publishing

This section described the high-level architecture for the iKnowBase Content Services.



Property	Description
Context Engine	Central controller that manages the flow of information within the various iKnowBase components.
Content Store	Stores the metadata and the information (Oracle database).
Adapters	Facilitates an easy integration with the external systems.
External systems	Systems that communicate data with iKnowBase through some standard adapters and vice versa. For example, systems such as dbs, other web applications, and web services.
Presentation Engines	Allows us to present the information to the end user in different meaningful formats.
Security Management	Provides access to a controlled environment. The user administration tool is part of Development Studio.

Metadata Management	Provides an environment for an easy management of the metadata in iKnowBase. Part of Development Studio.
Developer Desktop	Provides an environment for developing solutions on top of iKnowBase Content Services. Part of Development Studio.
Publishing Desktop	Provides an environment for publishing and administer the content on the end users dashboard. Also referred to as iKnowBase Content Studio.

6. Building iKnowBase Applications

This section describes briefly how you can build web applications using iKnowBase Content Services.

Key components

When you build applications, you typically expose a number of endpoints to the user:

- Pages that display information. These are typically implemented using Pages, Templates and various portlets.
- Pages that edit information. These are typically implemented using Pages, Templates and Forms.
- URLs that perform actions. These are typically implemented using Script Targets or PLSQL listeners.

Building Web Pages

Development of web pages using the iKnowBase Page and Template components together with iKnowBase Page Engine is based on the following patterns:

- Standalone pages
- Embedded pages
- Layout pages

Standalone Pages

Standalone pages are pages that contain all the required information to render a web page, within a single page definition. A standalone page is one page definition, one template, and one or more portlets. The templates can be developed by an HTML designer, while the actual page definition can be set up by a super user.

Perform the following steps to create a standalone page:

1. You create a Template, e.g. an HTML definition in which you define the layout of the page. The Template contains code that define the regions where you can place portlets (content components).
2. You create a Page, which you associate with the Template defined in the previous step.
3. Finally, you add portlets (menus, document lists, navigators, forms, etc) to be displayed in various regions on the page.

Standalone pages are simple to work with, but once you start building applications which require different layouts on different page, you often find yourself duplicating common elements such as headers, footers and menus. Therefore, standalone pages are generally best used during prototyping, and for pages with little or no layout and decorations.

Embedded pages

Pages can be re-used and embedded into other pages, including content and design. For example, you can create a page “my newsfeed” with multiple portlets, and then you can embed this page into both “front page” and “my page”.

Note that the output from the embedded page is included with all details into the embedding page. The embedded page should therefore not contain full html markup (such as the HTML and BODY tags), but only the actual information in a tag suitable for embedding (such as a DIV-tag).

Perform the following steps to embed pages:

1. Create a Template and a Page, and configure them to produce some sort of re-usable markup.
2. In the page where you want to embed the page, and use a **PageRunner** portlet to embed the content.

While embedding pages is a useful technique for structuring your page and portlet definitions, embedding by its very nature requires that the configuration is done on the outermost page. In some cases, it is possible that some parts of a collection of pages are common. For example, all pages in an

application follow the same layout. They have a banner at the top, and a menu on the left. One possible solution is that every page includes a banner and a menu in the right place. The disadvantage arises in that moment it is determined to make changes to the main layout. It then becomes necessary to go into all the pages that are in use, and modify these. A better solution for this scenario is to build and use a layout page.

Layout Pages

Layout pages are useful in situations where you build many pages that you want to “surround” with the same information. For example, you often want all of your pages to use the same banner and footer. Using layout pages, you can create a layout page that contains the surrounding decorations, and then design individual pages that refer to the layout page for decorations.

Perform the following steps to create and use a layout page:

1. Create a Template and a Page for the layout page. Add content that belongs to the layout page (such as banner, footer and menus). In addition, add a Template placeholder where the actual page content will appear.
2. Create a Template and a Page for your actual application page. Add content that belongs to the page, such as the relevant Viewers and Forms. In addition, configure the Page to use the layout page defined in the previous step.

When the end user accesses the application page, the iKnowBase Page Engine will detect that it should be enclosed in the layout page, and produce the complete page based on both the layout and the regular page. The advantage of this solution is that individual pages do not know anything about the overall layout, while the layout page does not know which or how many pages are actually based on this. In this case, the subpage knows in which page it is included.

Example template for a layout page:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" 'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>
<html>
<head>
  <title>xNet layout page</title>
  <style type="text/css">
    div { border:thin solid black; padding:5px; margin:5px; }
  </style>
</head>
<body>
  <div id="banner">
    ${page.regions.banner}
  </div>
  <div id="leftmenu">
    ${page.regions.leftmenu}
  </div>
  <div id="content" style="float:right; width:75%">
    ${page.clientPage}
  </div>
</body>
</html>
```

Adding Content to Web Pages

After you have built a web page using the iKnowBase Page and Template components, you can start adding portlets (content components) to it. You generally need to create and configure iKnowBase components before you can add and configure iKnowBase portlets to your web page, as the portlets depends on these components. See *iKnowBase Development Reference* for further information about iKnowBase components and portlets.

Tooltip functions

A tooltip function is used in viewers where you want to have mouseover functionality on title attributes.

They can be useful in lists of documents to display additional information about a document without navigating to detail pages.

Predefined tooltip functions

This is the list of tooltip functions available from a presentation style:

- Tooltip – cluetip – gist (used in search)
- Tooltip – cluetip – markup (used in search)
- Tooltip – cluetip – show preview page
- Tooltip – cluetip – snippets (used in search)
- Tooltip – cluetip – title/description

The three functions used in search will highlight the search criteria if entered by the user.

Define your own tooltip functions

To add new functions you can write your own PL/SQL-function, add it as a database command and use it in the presentation style layer. Use e.g. the function `ikb_tooltip.cluetip_preview` as an example.

Cluetip is the framework for tooltip functionalities and there are two different implementations offered:

- `ikb-cluetip-url` will use another url to extract the content that will be displayed as tooltip. Make sure the url (if its a iKnowBase page) is defined with a package alias in iKnowBase Development Studio. To create a new function with this method, the tooltip function has to return something like this:

```
class="ikb-cluetip-url" title="<title>" rel="/somurl/<docIdParam><docid>"
```

- `ikb-cluetip-title` is the simple usage of cluetip and will display a title and a description as cluetip without running additional pages. To create a new function with this method, the tooltip function has to return something like this:

```
class="ikb-cluetip-title" title="<title>|<description>"
```

Executing business logic

While pages are excellent tools for building pages that display information, they are often not ideal for implementing business actions. iKnowBase provides a number of tools that are better for this explicit purpose.

Script Targets

Script Targets are a mechanism used to attach scripts directly to a URL, where the application developer is free to add business logic as needed. In many ways, Script Targets match servlets in usage scenarios.

Script Targets are typically implemented using the Groovy script language, with iKnowBase providing access to the database and the Service API.

Perform the following steps to create a Script Target:

1. Create a Template of types "Action" and "Groovy Script". Add the business logic you want.
2. Create a Target. Refer to the Template you just created, and add a listen URL for the endpoint

7. Using Scripts and Templates

When building iKnowBase applications, you can use scripts and templates to implement logic and layout in a very efficient manner.

Inside iKnowBase, the term template is used to cover both scripts and templates. In fact, they are very much the same thing: A chunk of text which is evaluated at runtime, performing some kind of logic and producing some kind of output. You are free to implement templates in either FreeMarker, JavaScript or Groovy (where templates implemented using the latter two are typically called scripts).

Template concepts

The basic idea behind templates in iKnowBase can be summarized as follows:

- Execution starts inside an iKnowBase component declared in the DevelopmentStudio. This component first executes whatever duties it has, and then assembles a set of *model objects* for the template to use
- Control is then passed to the first template declared in the component. The template is then executed, and the output from the template is rendered in the context where the component is used

Available components

Templates are available for use with a large number of components:

- With a Page, the template is used to render the content of the page. The most important model objects are the portlets that are defined on the page.
- With a ContentViewer (through its Presentation style), the template is used to render the documents returned from the iKnowBase content store.
- With a DimensionViewer, the template is used to render the dimensions returned from the content store
- With a MenuViewer, the template is used to render the menu items returned from the content store
- With a TemplateViewer, the template is used to render arbitrary text, or run arbitrary logic.
- With a TemplateTarget, the template is used to render arbitrary text or run arbitrary logic.

Model objects

When an iKnowBase component passes control to a template, it also forwards information relevant to the execution. The actual objects available depends on the originating component, and the full list can be found in the API-reference documentation.

Available template languages

iKnowBase support template implementations in several languages:

- FreeMarker, an open source tool to generate text output based on templates.
- Groovy, a java-like scripting language.
- JavaScript (ECMAScript), implemented through the Nashorn script engine.
- HTML Script, the traditional HTML template format. Supports substitution tags (see below), as well as conditional execution using the configured parameter name and value.

The choice of language for a template is entirely up to you, but there are some typical guidelines:

- If your focus is on the *presentation* of output, FreeMarker is often the best choice.
- If your focus is on programming application logic, use one of the two other options. They are mostly equivalent, and the choice mostly comes down to what you know best: JavaScript programmers will chose the JavaScript engine, and java programmers will chose Groovy.
- If you need both application logic and complex logic, the best solution may be to use template chaining to combine either JavaScript or Groovy with FreeMarker, for the best of both worlds.

FreeMarker

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.

Note that Template-objects created in Development Studio have their own URL which can be used from other FreeMarker

templates, for #import and #include scenarios. The URL-syntax is

"urn:iknowbase:template:guid:<myguid>", which means

that use can use the syntax below in a FreeMarker template, to include another:

```
<#include "/urn:iknowbase:template:guid:9E23B9E919E73ECCE040000A18006DD1" >
```

Groovy

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.

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

When running in a web settings, where there is in fact an underlying HTTP servlet, the following objects are also available:

HttpServletRequest	A HttpServletRequest providing access to the underlying HttpServletRequest
HttpServletResponse	A HttpServletResponse providing access to the underlying HttpServletResponse

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 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 implicit `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 activity 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  
);
```

JavaScript

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

HTML script

You can use valid html. In addition, there is support for <ORACLE> tags to insert PL/SQL code.

Template chaining

iKnowBase allows *template chaining*, a mechanism where you combine multiple templates in one operation. For example, if you want to implement a search page, you may need to use a script (Groovy or JavaScript) to execute the actual search, but you still want the layout to be done using FreeMarker, which is better suited for that purpose.

Any component in iKnowBase that supports templating allows you to specify multiple templates, which you then name at will. iKnowBase will then proceed as follows:

- iKnowBase will find the first template defined in the list, regardless of name, and execute this template.
- If the template returns with a value, **and** this value is the name of another defined template, this template is executed next.

FreeMarker templates do not support returning a value, so any FreeMarker template will end the process. Groovy and JavaScript-templates, on the other hand, return the value of the last executed expression, so that it is simple to return the name of the next template:

```
// JavaScript template. Use a random value to choose template, then  
// evaluate expression returning either "view-full" or "view-minimal"  
var variant = Math.random() < 0.5 ? "full" : "minimal"  
"view-" + variant  
  
// Groovy-script  
def variant = System.currentTimeMillis() % 2 == 0 ? "even" : "odd"  
"view- $\$$ variant"
```

Error handling

All the template languages have their own, internal support for error handling (try-catch in JavaScript, try-catch in Groovy and #attempt/#recover in FreeMarker). However, it is often simpler to use the iKnowBase template error handling mechanism, which avoids the clutter inside each individual template.

If a template fails during execution, iKnowBase will look for a template named "ERROR". If this template is found, it will be executed. Simple!

8. Advanced Configuration

Domains

An iKnowBase installation may serve multiple uses at the same time. For example, it may serve an external site (<http://www.example.com>), an intranet (<http://intra.example.com>), a partner site (<https://partner.example.com>) and a customer service site (<https://service.example.com>).

An iKnowBase Domain defines properties that apply to a particular host name, to allow the user experience to differ between them.

The Default Domain

To avoid having to make a domain for every possible hostname:port combination, you can designate a domain as the default domain. The default domain will apply to all requests that do not match any other domain.

Note that having a default domain is required for sending e-mail, since most e-mail functionality is unrelated to a user request, and hence it is not possible to find the proper domain in any other way.

Custom Access Control

For a domain you may override or extend the default access control for content modification and deletion. For example, for an integrated solution you can check the metadata in the external document before permitting any modification.

The custom access control must be implemented as a PL/SQL procedure that must be specified per domain.

The signature of the plug-in is:

```
CREATE OR REPLACE PROCEDURE custom_access_control (  
    site_id          IN     NUMBER,  
    document_id     IN     NUMBER,  
    user_id         IN     NUMBER,  
    modify          IN OUT NUMBER,  
    delete          IN OUT NUMBER  
)
```

Parameter	Description
site_id	Not used
document_id	Document ID of document to be checked
user_id	User ID of user making the request
modify	Return 0 to disallow modifications; 1 to allow.
delete	Return 0 to disallow delete; 1 to allow.

Public user

For public sites, such as the typical internet site on www.example.com, clients are not logged on. In order to be able to perform the required access control, iKnowBase require you to specify a “public user” which is used for clients which are not logged on. By giving this user access to the required public documents, public (not logged on) clients will also see this content.

Further, sometimes you want to make sure that all users always see the same image of a site. For example, when searching on the www.example.com site, you do not want logged on users to find their private content from the intranet. By selecting “always run as this user” for the corresponding domain, every request will look like a public (not logged on) request.

Weight on free-text search

iKnowBase utilizes the advanced Oracle Text search engine. This engine lets you specify different weight to different search criteria, so that certain search hits are prioritized over others.

The domain properties let you specify the search criteria weights used for searches originating at the specified domain.

Use of search weights is for advanced setups. In general, higher values give higher weight.

Attribute Security Function

The attribute security function enables you to implement logic which decides whether document attributes are savable or just visible. The attribute security function is called upon opening an existing document in an iKnowBase Form and saving an existing document, both through a Form, through the WebDAV service and through the Service API.

When the end user opens an existing document in an iKnowBase Form, read-only document attributes are displayed, but are not available for update. The status of a document may change after the end user has opened in an iKnowBase Form, and document attributes may have become read-only even though available for update in the iKnowBase Form. When the end user saves an existing document from iKnowBase Form, changes to read-only document attributes are ignored.

If an end user tries to save an Office document to an iKnowBase document for which the file document attribute is read-only, he gets an error message.

If a method in the iKnowBase Service API is used for updating an existing iKnowBase document, the save operation will fail if one of the read-only document attributes is changed. In this situation, an error message is logged to the IKB_ERROR_TAB table.

Enable and Disable the Attribute Security Function

You enable the attribute security function by setting the `attribute_security_function` constant to the name of the attribute security function in the `IKB_GLOBAL_PREFS` database package. Example:

```
attribute_security_function CONSTANT varchar2(92) :=  
  'iknowbase.sample_attribute_security';
```

You disable the attribute security function by setting the `attribute_security_function` constant to empty string in the `IKB_GLOBAL_PREFS` database package:

```
attribute_security_function CONSTANT varchar2(92) := '';
```

Note: The `attribute_security_function` constant must be declared and initialized.

Signature of the Attribute Security Function

The document attribute rules should be implemented in the database function as specified by the `attribute_security_function` constant.

It should follow the following signature:

```
FUNCTION sample_attribute_security (p_params ot_attribute_security)  
RETURN ct_secured_attributes;
```

Parameter	Description
<code>p_params</code>	Information about the attribute security request, containing information like the document id, parent document id, user id, id of the iKnowBase Form, and document type id.
RETURN	An array of attributes that should be read-only. Note: The attribute <code>ot_secured_attribute.document_reference</code> is only for internal use, and you are not required to assign a value to it.

Run custom access control

Custom access control enable you to define an additional control to the default iKnowBase access control when validating if a user has read access to a document or not. If enabled, a customizable function

is added to all queries used for this type validation. Be aware if enabled it will be executed across all viewers and domains. The only place you can override it is by adding a custom access control at viewer definition level.

Enable and Disable the custom access control

You enable custom access control by setting the `run_custom_access_control` constant to `TRUE` in the `IKB_GLOBAL_PREFS` database package. When set to true, a conditional compile is executed and the custom function is enabled for the schema. Example:

```
run_custom_access_control CONSTANT boolean := TRUE;
```

You disable the custom access control by setting the `run_custom_access_control` constant to `FALSE` in the `IKB_GLOBAL_PREFS` database package:

```
run_custom_access_control CONSTANT boolean := FALSE;
```

Note: `run_custom_access_control` must be declared and initialized. By default, an empty function is provided with return code = 0.

Signature of run_custom_access_control

It has the following signature:

```
run_custom_access_control (
    p_document_id    IN          NUMBER,
    p_user_id        IN          NUMBER,
    p_acl_guid       IN          RAW,
    p_acl_id         IN          NUMBER,
    p_owner_id      IN          NUMBER) return
number;
```

Parameter	Description
p_document_id	Document id
p_user_id	User identifier in iKnowBase
p_acl_guid	The GUID of the ACL if the document has an ACL
p_acl_id	The ID of the ACL if the document has an ACL
p_owner_id	The Owner ID of the document
RETURN	1 will give access even if the user has no access to the document, 0 means the user must have ordinary access to the document

Default common scripts configuration

This section describes supported installation property options for the `context.commonScripts` directive.

jQuery

Property name	Value
<code>com.iknowbase.commonScripts.jqueryMigrateReference</code>	URN or URL of the jquery script
<code>com.iknowbase.commonScripts.jqueryMigrateRefer</code>	URN or URL of the jquery-migrate script

Examples jQueryReference:

Value	Description
Empty value	Default jQuery version
<code>urn:iknowbase:jquery:1.8</code>	jQuery version 1.8.x (unspecified bugfix version)
<code>urn:iknowbase:jquery:1</code>	jQuery version 1.x.x (unspecified minor and bugfix version)

urn:iknowbase:jquery:2	jQuery version 2.x.x (unspecified minor and bugfix version)
/ressurs/customer/jquery.js	Customer specific jQuery (use URL)
//code.jquery.com/jquery-1.11.3.min.js	External jQuery from CDN (use URL)

Examples jQueryMigrateReference:

Value	Description
Empty value	No jquery-migrate script
urn:iknowbase:jquery-migrate	jQuery migrate (unspecified major, minor and bugfix version)
/ressurs/customer/jquery-migrate.js	Customer specific jQuery migrate (use URL)
//code.jquery.com/jquery-migrate-1.2.1.min.js	External jQuery migrate from CDN (use URL)

9. Using Apache Solr for Content Search

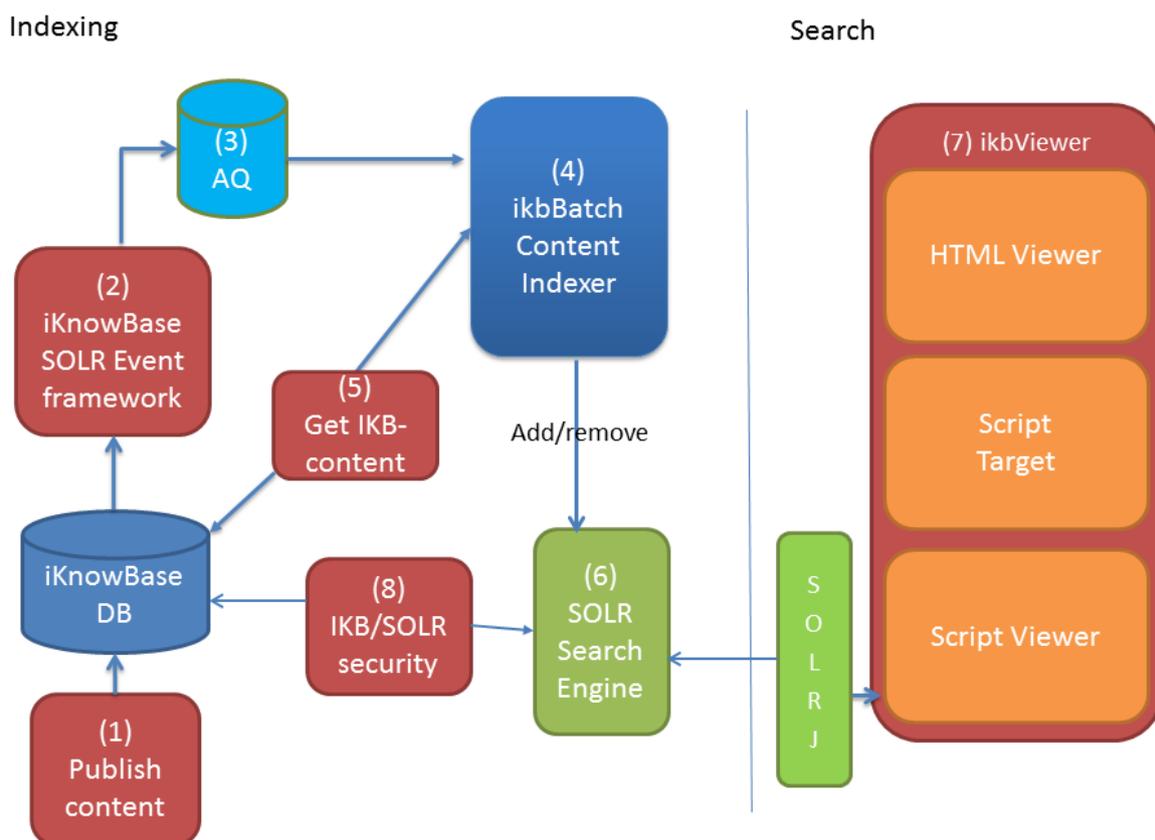
Concept

iKnowBase comes with ready-to-use components for integration with the Apache Solr open source enterprise search platform (<http://lucene.apache.org/solr/>).

When using iKnowBase together with Apache Solr, the following components will be in use:

- The Apache Solr server is installed on one or more computers. Inside Apache Solr, an iKnowBase search component is installed to handle security considerations
- The database repository and ikbBatch application cooperates to send index updates from iKnowBase to Apache Solr
- The ikbViewer application has user interface components to easily generate search requests and navigate the search result

The process for indexing works somewhat like this:



- The content is inserted, updated or deleted in iKnowBase (1). This can be done thru Forms, Service API or any other user interface made to update content in iKnowBase.
- The document may be triggered by a Solr event if the event condition matches the document (2).
- If triggered, an "index request" is queued to the Oracle Database AQ-system (3). The operation will either be update or delete.
- The ikbBatch application listens to index requests (4).
- If the document is new or updated, ikbBatch will retrieve the document (5), as described in the Solr Configuration. The Solr Configuration defines how the document should be represented in Apache Solr and maps the document/attribute values to Solr fields.

- Finally, `ikbBatch` creates and sends “update” or “delete” messages to Apache Solr for updating the Solr index (6).

The process for search works somewhat like this:

- Whenever a user initiates a search operation (7), the user interface component creates a `SolrQuery` and submits to an `iKnowBase SolrSearchClient` component for execution.
- The search client adds security related information to the request, signs the request using a secure Message Authentication Code, and sends the request to Apache Solr.
- Inside Apache Solr, the `iKnowBase` search component verifies the security information, and adds the required search conditions (8).
- Apache Solr returns a normal search response which is rendered by the user interface components in `iKnowBase`.
- A result set from Solr can easily be merged with a viewer. This will enhance the final result and you don't have to store everything in Solr.

Installation and setup

The *Installation Guide* contains installation guidelines. You will first need to configure the Solr server itself, and then you will need to configure the connections from `iKnowBase` to Solr. After you're done with the installation you are ready to index your content.

iKnowBase security

The Apache Solr Search Server, which is distributed together with `iKnowBase`, includes `iKnowBase Solr` components which handle security in terms of authorized access to documents. The `iKnowBase` security search component is configured in the configuration file “`solrconfig.xml`”, and configured for use by the search handler “`/select`”. Search handlers, which use this component, will load security information from `iKnowBase` and filter the result set by `iKnowBase` access control lists.

Note: If you will configure new search handlers, include the `iKnowBase` security search component to ensure authorized access to `iKnowBase` data. For information regarding security in autocomplete operations, see *Configuring search suggestions* below.

Configuring the indexing process

Before you start indexing your content you need to decide the following:

- What kind of documents/information should be indexed? You need to investigate your content and define what to index. When you have defined all indexable content you will need to represent the set in one or more Indexing events in Development Studio (available under the Advanced tab), see *Development Reference*. An Indexing event will trigger all documents with the given condition and issue an update statement to the Content Indexer.
- When you have decided which documents to be indexed, you then have to define what to index within a document. You must create a Solr configuration in Development Studio (available under the Advanced tab), where you define all the attributes to be indexed and how they should be represented in Apache Solr, see *Development Reference*. Should the attribute be indexed itself? Should we be able to search for the attribute value text when we do a freetext search? Should it be possible to display the value in the result set? The Solr configuration screen will help you by presenting the most common attributes in your database, they are normally good candidates for indexing.

Building a search page

To build a search page, you will use a Groovy-based template (Template Viewer or Script Target), where the `iKnowBase SolrSearchClient` component provides access to the Solrj library.

The basic flow for a search page is as follows, see below for examples:

- Acquire a search client from the available beans
- Acquire a `SolrQuery` using the search client. The search client can either provide a new `SolrQuery` every time, or provide a `SolrQuery` which is stored in the user session and can be reused. Using a session-based `SolrQuery` avoids having to send all query configuration on the URL, as the current state is already present.

- Use the search client to apply URL-parameters to the SolrQuery, or set up the required parameters manually

Create unique package- and class names in every template to avoid conflicts with other solr templates. Use a descriptive name and avoid using com.iknowbase.

- The package declaration defines where the objects declared in the file are stored, e.g “facets”, “sortFields” og “SolrjSearchClass”.
- The “searchClients.default” defines the Solr search instance to be used (Defined in Solr Configuration)

```
package no.customer.intranet;
import org.apache.solr.client.solrj.SolrQuery;
def facets = [ "type_${context.language}", "status_${context.language}",
  "title"];
def sortFields = [ "document_id", "type_${context.language}",
  "updated_index_store_date" ];
new SolrjSearchClass (searchClients.default, html, context, facets,
  sortFields).run();
```

To merge a result set with a iKnowBase Solr Viewer you first need to define a Solr viewer with a Solr presentation style. Then in the Groovy based template, you combine like this :

```
def rowset = this.searchClient.getRowSet(response, "<External key to the Solr
  Viewer>");
for (int i=0 ; i<response.results.size() ; i++) {
  renderDocument (response.results[i], rowset.rows[i]);
}
```

In renderDocument, you will have access to data from both Solr and the Viewer. e.g.

```
def renderDocument(document, ikbRow) {
ul {
  li("description=${document.description}");
  li("ikb.title=${ikbRow.document.title}");
  ikbRow.items.each { key, item ->
    li {
      mkp.yield(key + ": ");
      mkp.yieldUnescaped (item.getAsString() ?: "")
    }
  }
}
}
```

In terms of documentation, you will be interested in the following:

- The *Solrj javadoc*
- The *iKnowBase PresentationServices API*, in particular the *javadoc for SolrSearchClient*

Sample search page structure

A transport set is provided as an example where a basic search page is included with faceting, ordering and autocomplete based on a facet search. Import the file

etc/EXP-IKB_MASTER_71-F250F353F65159F3E040000A18007A88-iKnowBase-Demo-SolrExample.dmp from /ikbStudio/advanced/importjobs. It contains the “essentials” for using a search client in the page /demo/solr.

Configuring search suggestions

Search suggestions (autocomplete) can be implemented in several ways.

On the provided sample page, see chapter *Sample search page structure*, autocomplete is implemented using a faceted search. On the search page there is an Ext JS combobox which loads data through an iKnowBase script target. The script target forwards an autocomplete request, which actually is a faceted

search, to Solr. **Note:** The property facetMinCount is set to 1 to prevent unauthorized access to data, ie. a facet is only returned if it is in use in one of the documents available to the user.

The Solr Suggester component provides an alternative way to implement autocomplete. **Note:** The Solr Suggester provides no mechanism to add a security filter and the user may get access to unauthorized data, ie. if the completion comes from the title index, the user may see a document title even though he is not authorized to view the document.

Monitoring the Solr solution

A few key items to check:

- On the ikbBatch page, check that the content indexing queue is emptied.
- Monitor the solr-instance using the console at `http://<hostname>:<solr-port>/solr/#/`
- Optimize the solr index using the console at `http://<hostname>:<solr-port>/solr/#/~cores/<corename>`

10. Using iKnowBase Instant

Concept

iKnowBase comes with client and server side support for creating applications with real time asynchronous messaging.

Features of iKnowBase Instant:

- Exchange messages between clients
- Subscribing web clients will receive events as soon as they occur.
- Both server and client can push messages.
- Multiple message topics for exchanging messages.
- Public (all) and private (recipient only) messages on chosen topic.
- See who is connected to a topic and who joins and leaves the topic.
- Multiple transport mechanisms (Long Polling and WebSocket) with fallback to secondary transport mechanism.
- JavaScript API for publishing and consuming messages in real time, see *Javascript library > iKnowBase.Instant* in the *iKnowBase API Reference*.
- PLSQL API and HTTP API for publishing messages, see *Instant Server > PLSQL API* and *Instant Server > HTTP API* in the *iKnowBase API Reference*.

The key feature with iKnowBase Instant is that a web client will get notified as soon as a message matching the subscription is available. The message may be whatever the applications use the message topic for (new document added, document updated, user has logged on to instant, “Hi! How are you?”, ...). This enables the type of rich interaction possibilities for web applications that has previously only been available in traditional non-web applications.

JavaScript API and transport mechanisms

The JavaScript API supports publishing and consuming messages in real time and is the primary API available for iKnowBase Instant. The API and corresponding server side support in the Instant Server is based on the *Atmosphere Framework*.

The Instant Server supports multiple asynchronous transport mechanisms, discussed in the following chapters.

Synchronous HTTP

Traditional synchronous HTTP transport follows the following pattern:

1. Client sends a HTTP request to the server and waits for completion.
2. The server generates a HTTP response and replies to the client.
 1. The client receives the HTTP response and has completed the message exchange. The network connection is closed or, if HTTP KeepAlive is in use, kept open for a period of time for subsequent requests.

The client will not get notified if an event occurs after the message exchange has completed. The client needs to send a new request if it wants new content.

Ajax (Asynchronous JavaScript and XML) techniques can be used by the client to make the application behave in a rich asynchronous way, but the the client still needs to poll the server for new content.

Asynchronous HTTP

Asynchronous HTTP transport is preferred for applications where you want to maintain a dialog between client and server or where the server facilitates a dialog between clients.

Asynchronous HTTP transport follows the following pattern:

1. Client subscribes to messages from the server and has callback functions in place in case an event occurs.
 1. Control is immediately returned to the client.

2. An event is detected by the server and is sent to the client.
 1. The client processes the event in the relevant callback function.
3. Another event is detected by the server and is sent to the client.
 1. The client processes the event in the relevant callback function.
4. The client sends a message to the server.
 1. The server processes the message
5. ...

The client will get notified as soon as an event occurs.

While the client still uses Ajax (Asynchronous JavaScript and XML) techniques for subscribing and processing messages, the events are sent to the client as soon as they are available.

long-polling

The long-polling technique, also known as Comet, is a variant of the traditional HTTP request polling where the server waits until a message is available or timeout occurs before returning the response. The client will immediately re-connect and have a connection available for more messages.

long-polling is supported by all web browsers and is the default transport mechanism configured in the Instant JavaScript client library.

websocket

Websocket provides a bi-directional communication channel over a single TCP connection. It uses HTTP for handshake only and then upgrades to websocket protocol. Both client and server may push messages using the established communication channel without reestablishing the channel between messages.

The use of websocket will significantly improve performance when sending and receiving messages as all messages in a conversation is sent/received using the same connection. The overhead of the HTTP protocol is also not added to every message, reducing the amount of network traffic. Although more performant, the protocol is currently not as widely supported as long-polling.

Websocket with iKnowBase Instant is supported on:

Web browser	Minimum version
Chrome	13
Firefox	9
Internet Explorer	10
Edge	20
Opera	11
Safari	5
Safari (iOS)	5

Application Server	Minimum version	Comment
iKnowBase web server	6.3	The first release including iKnowBase Instant

Note#1: The protocol must also be supported by all intermediate middleware servers between the client and application server, such as load balancers, caches and front web servers. Take care to examine your web clients and middleware infrastructure for protocol and supported number of connections to see if you can use websocket.

Note#2: It is possible to deploy the Instant Server server separate from the other iKnowBase application when using a load balancer or when Cross Origin Resource Sharing (CORS) is enabled. This brings websocket availability to otherwise unsupported middleware infrastructure.

Transport failover

The JavaScript API offers an option “fallbackTransport” for configuring transport to use in case the primary transport fails. fallbackTransport defaults to long-polling.

Example:

Transport: websocket

FallbackTransport: long-polling

- Firefox supports websocket and connects with websocket
- If intermediate server rejects websocket connection (unsupported), Firefox uses fallback to long-polling.
- IE9 does not understand websocket and connects with long-polling

Note#1: Some intermediate servers may hog or drop connections that use an unsupported transport instead of rejecting them. This may prevent the client from using fallbackTransport within an acceptable period of time.

Blocking and non-blocking I/O

Web and application servers have traditionally used a blocking I/O model when handling web requests. The server thread processing the request will be occupied until the response has been delivered. Supporting a large amount of asynchronous clients who are “always connected” would relatively quickly use all available server threads and block the server for incoming requests.

Servers supporting non-blocking I/O will queue the request as soon as it is received and free the request acceptor thread for new requests. The request itself is further processed/used at some later time by the server. This allows for a huge amount of simultaneous connected clients compared to blocking I/O.

Servers, both application and intermediate servers, participating in the asynchronous support (for JS API clients) in iKnowBase Instant should always use non-blocking I/O.

Topics

A topic is an area where clients exchange messages. A client can only subscribe to one topic per connection, but the client can use multiple connection if needed.

The topic identifier is composed of:

- Name (required)
- Topic options (optional)

Two topics with same name and different topic options are two different topics.

See *Topic name and topic options* in the *iKnowBase API Reference* for rules and options.

There are two types of topics available:

- InstantTopicImpl (default)
- InstantTopicUserListImpl

InstantTopicUserListImpl supports all functionality in InstantTopicImpl and adds userList support. See chapter below for further description about userList.

Message formats

Two types of message formats are supported:

- TEXT (default)
- IKB

The TEXT message format is a plain text topic where the clients send and receive text messages (or anything coded as text). The message does not contain who sent the message or who the message is for, unless the clients add that information themselves.

The IKB message format adds structure to the messages and adds support for detecting who sent the message, who the message is for (private messages) and a message type. See *Topic name and topic options* in the *iKnowBase API Reference* for details regarding the IKB message format for requests and responses.

IKB message format is mandatory on userList (InstantTopicUserListImpl) topics.

Public and private messages

For the TEXT message format, all messages are public.

For the IKB message format, messages can be sent to a specific user (private message) by setting the recipients username in the toUserName property of the message.

Example public message in IKB-format sent by IKBUSER1:

```
Request
{
  "toUserName": ""
  , "messageType": "USER_MSG"
  , "data": "This is a public message"
}
Response
{
  "toUserName": ""
  , "fromUser": {
    "guid": "9BEA12EB56126795E040000A180038B7"
    , "id": 1111
    , "username": "IKBUSER1"
    , "dn": "IKBUSER1"
    , "label": "IKB User1"
  }
  , "messageType": "USER_MSG"
  , "correlationId": null
  , "data": "This is a public message"
}
```

Example public message in IKB-format sent by IKBUSER1 to IKBUSER2 :

```
Request
{
  "toUserName": "IKBUSER2"
  , "messageType": "USER_MSG"
  , "data": "This is a public message"
}
Response
{
  "toUserName": "IKBUSER2"
  , "fromUser": {
    "guid": "9BEA12EB56126795E040000A180038B7"
    , "id": 1111
    , "username": "IKBUSER1"
    , "dn": "IKBUSER1"
    , "label": "IKB User1"
  }
  , "messageType": "USER_MSG"
  , "correlationId": null
  , "data": "This is a public message"
}
```

User list support

The server will keep a list of all connected users to a topic if the topic is configured with option "userList=true". A user is present at the topic if the user has one or more active connections.

When user list support is activated, the clients may use any of the following subscription *options*:

- requestUserList
- subscribeUserListChanges

The Instant APIs also support server requests providing on demand alternatives for these subscription options, see

- *Javascript library* > *iKnowBase.Instant*
- *Instant HTTP API*
- *_Instant PLSQL API*

requestUserList

If the client subscribes with requestUserList, the client will receive a list of all users connected to the topic after the connection has joined. The returned list contains objects of type UserReference, which contains guid, id, username, dn and label, where label is the user's "<firstname_if_any> <lastname>" as registered in iKnowBase User Directory.

The user list message is a private message sent to the user's connection that joined the topic.

Example message payload in IKB-format for requestUserList for user IKBUSER1 on a topic where IKBUSER2 was also present:

```
{
  "toUserName": "IKBUSER1"
  , "fromUser": {
    "guid": null
    , "id": 0
    , "username": "ikb$instant"
    , "dn": null
    , "label": "iKnowBase Instant Server"
  }
  , "messageType": "IKB.USERLIST.USERS"
  , "correlationId": null
  , "data": [
    {
      "guid": "9BEA12EB56126795E040000A180038B7"
      , "id": 1111
      , "username": "IKBUSER1"
      , "dn": "IKBUSER1"
      , "label": "IKB User1"
    }
    , {
      "guid": "9582B1E8B4AA4BDA9B7E0B6422A1D1F7"
      , "id": 2222
      , "username": "IKBUSER2"
      , "dn": "IKBUSER2"
      , "label": "IKB User2"
    }
  ]
}
```

subscribeUserListChanges

If the client subscribes with subscribeUserListChanges, the client will receive join and leave messages when a user joins or leaves the topic.

Join/leave messages are public messages and are sent to all users on the topic who subscribe to user list changes.

Example message payload in IKB-format for subscribeUserListChanges JOIN for user IKBUSER1:

```
{
  "toUserName": null
  , "fromUser": {
    "guid": null
    , "id": 0
  }
}
```

```

    , "username": "ikb$instant"
    , "dn": null
    , "label": "iKnowBase Instant Server"
  }
, "messageType": "IKB.USERLIST.JOIN"
, "correlationId": null
, "data": {
  "guid": "9BEA12EB56126795E040000A180038B7"
  , "id": 1111
  , "username": "IKBUSER1"
  , "dn": "IKBUSER1"
  , "label": "IKB User1"
}
}

```

Example message payload in IKB-format for subscribeUserListChanges LEAVE for user IKBUSER1:

```

{
  "toUserName": null
  , "fromUser": {
    "guid": null
    , "id": 0
    , "username": "ikb$instant"
    , "dn": null
    , "label": "iKnowBase Instant Server"
  }
  , "messageType": "IKB.USERLIST.LEAVE"
  , "correlationId": null
  , "data": {
    "guid": "9BEA12EB56126795E040000A180038B7"
    , "id": 1111
    , "username": "IKBUSER1"
    , "dn": "IKBUSER1"
    , "label": "IKB User1"
  }
}

```

APIs

The APIs are documented in the *iKnowBase API Reference*:

- JavaScript API for publishing and consuming, see *Javascript library > iKnowBase.Instant*
- HTTP API for publishing, see *Instant Server > HTTP API*
- PLSQL API for publishing, see: *Instant Server > PLSQL API*

Installation and setup

See the *iKnowBase Installation Guide*.

Deployment and scaling

The Instant Server does not support application server clustering and a specific topic is required to exist in one JVM only to enable communication between clients on that topic. Clients can only communicate with other clients on the same JVM.

To support more traffic / connections than one server can handle, the topics can be divided among multiple servers:

- Server 1: Handles topics /Topic1, /Topic2 and /Topic3
- Server 2: Handles topics /Topic4

The routing can be done by a load balancer with URL routing capabilities, as the topic name is part of the HTTP Request URL. This routing will be transparent for the connected clients.

An alternative to routing via load balancer is to use Cross Origin Resource Sharing (CORS) support, enabling clients to connect directly to a server host:port different from where the web page was loaded. The CORS address MUST match the protocol used on the webpage. If the web page uses HTTPS, then the CORS address must also be HTTPS.

NOTE: Only one Instant Server can consume messages from the PLSQL API. Enable/disable with Installation Property `com.iknowbase.instant.aq.enabled`.

Security

Authentication

The Instant Server offers protected services that requires client authentication.

Authentication to the Instant Server can be accomplished through either

1. Use Secure Token (recommended) or
2. Direct application container authentication

Once the client connection has been authenticated, the Instant Server will store the user's identity for that specific connection as long as the connection is alive.

If the client tries to subscribe to a service that requires authentication before the client has authenticated, the subscribing request will receive a "HTTP 401 Unauthorized" response and the asynchronous connection will be terminated.

Authenticating with Secure Token

A user that is already authenticated to the `/ikbViewer` application can get a Secure Token (use Freemarker or Groovy support) and submit this when connecting to `/ikbInstant`. `/ikbInstant` will verify the token and retrieve the user's identity without any user interaction.

To use Secure Token Authentication:

1. Configure the *SecureTokenEngine*, see *iKnowBase Installation Guide*.
2. Add the user's Secure Token to the Instant subscribe request. (Available when using Freemarker and Groovy)
3. Use default Instant servletURL `/ikbInstant/ikbInstant`

Authenticating with direct application container authentication

Authentication can also be done without Secure Token support. The container is responsible for authenticating the request and may require user interaction.

To use direct application container authentication:

1. Use protected Instant servletURL `/ikbInstant/private/ikbInstant`

Authorization

Feature	Authentication	Required privilege	Comment
No topic options on public servlet URL	NOT NEEDED	Not applicable	
No topic options on private servlet URL	REQUIRED	Authenticated	Used for direct application container authentication support
<code>messageFormat=IKB</code> topic option	BENEFITS	Not applicable	Authentication enables usage of <code>fromUser</code> and <code>toUserName</code>
<code>userList=true</code> topic option	REQUIRED	Authenticated	Only authenticated users can access a <code>userList</code> topic
<code>ikb\$console</code>	REQUIRED	Authenticated and administrator	Administration and monitoring

Private messages

A message can be targeted to a specific user, identified by the user's username. The Instant Server guarantees that the message is only delivered to connections authenticated with that username.

This service requires that

1. The topic is configured with the "IKB" message format.
2. The publisher includes the recipient's username in the "toUserName" property of the message.
3. The recipient must be authenticated and connected for the message to be delivered.

Monitoring

Standard iKnowBase monitoring service `ikb$console` is available for the application. It is equipped with a separate Instant tab with information about topics, options, users and connection information.

Building solutions

See *iKnowBase API Reference*:

- JavaScript API for publishing and consuming, see *Javascript library > iKnowBase.Instant*
- HTTP API for publishing, see *Instant Server > HTTP API*
- PLSQL API for publishing, see: *Instant Server > PLSQL API*

Project wide settings

The following code is typically added to a template or a reusable page component. This includes the necessary JavaScript and sets default options avoiding code copy on every Instant enabled page.

Project wide sample basic

```
<script src="/ressurs/iknowbase/iknowbase-instant-min.js"></script>
<script>
// Set project-wide defaults; typically added in layout template with html-
header
iKnowBase.Instant.setDefaultOptions({
  subscriptionOptions: {
    // Not using any custom options
  },
  atmosphereOptions: {
    // Using a project specific onOpen function. Optional.
    onOpen: function (response) { iKnowBase.log ("request.onOpen
(global)", response); }
  }
});
```

Project wide sample with secure token authentication

```
<script src="/ressurs/iknowbase/iknowbase-instant-min.js"></script>

[#ftl]
[#if context.user.isLoggedOn]
  [#assign secureTokenAuth = "'_ikbUserToken':" + "'" +
  context.user.token.value + "'"]
[#else]
  [#assign secureTokenAuth = '']
[#/if]

<script>
// Set project-wide defaults; typically added in layout template with html-
header
iKnowBase.Instant.setDefaultOptions({
  subscriptionOptions: {
    ${secureTokenAuth} // Authentication header from ikbViewer
```

```

    },
    atmosphereOptions: {
        // Using a project specific onOpen function. Optional.
        onOpen: function (response) { iKnowBase.log ("request.onOpen
(global)", response); }
    }
});

```

Project wide sample with secure token authentication and CORS enabled

CORS configuration can be enabled in project wide settings.

NOTE: CORS must be enabled in server configuration as well, see *InstantServerConfiguration* CORS option.

```

<script src="/ressurs/iknowbase/iknowbase-instant-min.js"></script>

[#ftl]
[#if context.user.isLoggedOn]
    [#assign secureTokenAuth = "'_ikbUserToken':" + "'" +
context.user.token.value + "'"]
[#else]
    [#assign secureTokenAuth = '']
[#/if]

<script>
// Set project-wide defaults; typically added in layout template with html-
header
iKnowBase.Instant.setDefaultOptions({
    servletURL: "http://CORS_SERVER_NAME:CORS_SERVER_PORT/ikbInstant/
ikbInstant", //CORS: URL to servlet - the page itself should be served
through a different URL
    subscriptionOptions: {
        ${secureTokenAuth} // Authentication header from ikbViewer
    },
    atmosphereOptions: {
        enableXDR: true, // Enable CORS
        // Using a project specific onOpen function. Optional.
        onOpen: function (response) { iKnowBase.log ("request.onOpen
(global)", response); }
    }
});

```

Instant subscribe with CORS

CORS configuration can be enabled during subscribe.

NOTE: CORS must be enabled in server configuration as well, see *InstantServerConfiguration* CORS option.

```

iKnowBase.Instant.Atmosphere.subscribe({
    servletURL: "http://CORS_SERVER_NAME:CORS_SERVER_PORT/ikbInstant/
ikbInstant", //CORS: URL to servlet - the page itself should be served
through a different URL
    topicName: "/Topic1",
    subscriptionOptions: {
        // not using any custom options
    },
    atmosphereOptions: {
        enableXDR: true, // Enable CORS
        onMessage: onMessage // override onMessage
    }
}

```

```
});
```

Simple Web to Web Client using text messages

This sample enables web clients to send and receive messages on the specified topic using `messageFormat=TEXT`.

Prerequisite: Included project wide settings basic (see *Building solutions > Project wide settings basic*).

```
// Page-specific application script
(function () {

    var topicName = "/Topic1"; // Using default
    messageFormat=TEXT for this topic
    var channel; // Reference to topic
    connection. Used for publish and close.

    function onMessage(response) {
        iKnowBase.log ("MySample.onMessage", response);
        if (response.status == 200) { // HTTP status code=200 means normal
            response.
            $('#messages ul').prepend($('- </li>').text("Text message
            received: " + response.responseBody));
        }else{
            iKnowBase.log('ERROR: Expected HTTP status code 200, but got HTTP
            status=' + response.status, response);
        }
    }

    function subscribe() {
        channel = iKnowBase.Instant.Atmosphere.subscribe({
            topicName: topicName,
            subscriptionOptions: {
                // Not using any custom options
            },
            atmosphereOptions: {
                onMessage: onMessage // Use page specific onMessage callback
            }
        });
    }

    function sendMessage(){
        var message = jQuery("#messageToSend").val(); // Retrieve the message
        iKnowBase.log("sendMessage: " + message); // Log the message
        channel.push({data: message}); // Send the message to
        topic
        jQuery("#messageToSend").val(""); // Clear the input
        message element
    }

    window.MySample = {
        subscribe: subscribe,
        sendMessage: sendMessage
    }

})();

MySample.subscribe(); // Subscribe to topic

</script>

<div>

```

```

    <h4>type your message here:</h4>
    <input id='messageToSend' type='text' /><button id='sendMessage'
name='sendMessage' type='button' onClick='MySample.sendMessage()'>Send
message</button>
</div>

<div id="messages">
    <h4>Topic messages</h4>
    <ul>
    </ul>
</div>

```

Simple Web to Web Client using IKB messages and secure token authentication

This sample enables web clients to send and receive messages on the specified topic using messageFormat=IKB.

Prerequisite: Included project wide settings with authentication (see *Building solutions > Project wide sample with secure token authentication*).

```

// Page-specific application script
(function () {

    var topicName = "/Topic1?messageFormat=IKB"; // Using messageFormat=IKB
for this topic
    var channel; // Reference to topic
connection. Used for publish and close.

    function onMessage(response) {
        iKnowBase.log ("MySample.onMessage", response);
        if (response.status == 200) { // HTTP status code=200 means normal
response.
            try {
                // First decode the message_format=IKB wrapping as JSON
                var ikbMsg = jQuery.parseJSON(response.responseBody);
            } catch (e) {
                iKnowBase.log('This does not look like a valid JSON:
',response.responseBody);
                return;
            }

            $('#messages ul').prepend($('<li></li>').text("IKB message received:
" +
                                                                    "fromUser=" +
((ikbMsg.fromUser)?ikbMsg.fromUser.label:null) +
                                                                    ", toUserName=" +
ikbMsg.toUserName +
                                                                    ", messageType=" +
ikbMsg.messageType +
                                                                    ", data=" +
ikbMsg.data));
            }else{
                iKnowBase.log('ERROR: Expected HTTP status code 200, but got HTTP
status=' + response.status, response);
            }
        }

        function subscribe() {
            channel = iKnowBase.Instant.Atmosphere.subscribe({
                topicName: topicName,

```

```

        subscriptionOptions: {
            // Not using any custom options
            // Authentication was added in project wide settings, but
could also be added here
        },
        atmosphereOptions: {
            onMessage: onMessage // Use page specific onMessage callback
        }
    });
}

function sendMessage(){
    var message = jQuery("#messageToSend").val(); // Retrieve the message
    iKnowBase.log("sendMessage: " + message);     // Log the message
    channel.push({                                // Send the message to
topic
        toUserName: ''
        , messageType: 'SAMPLE_MESSAGE'
        , data: message
    });
    jQuery("#messageToSend").val("");           // Clear the input
message element
}

window.MySample = {
    subscribe: subscribe,
    sendMessage: sendMessage
}

}());

MySample.subscribe(); // Subscribe to topic

</script>

<div>
    <h4>type your message here:</h4>
    <input id='messageToSend' type='text' /><button id='sendMessage'
name='sendMessage' type='button' onClick='MySample.sendMessage()'>Send
message</button>
</div>

<div id="messages">
    <h4>Topic messages</h4>
    <ul>
    </ul>
</div>

```

Chat With User List Support

This sample shows a simple chat client with userList support.

Prerequisite: Included project wide settings with authentication (see *Building solutions > Project wide sample with secure token authentication*).

```

// Page-specific application script
(function () {

    var topicName = "/Topic1?messageFormat=IKB&userList=true"; // Using
messageFormat=IKB for this topic

```

```

var channel; // Reference
to topic connection. Used for publish and close.

function onMessage(response) {
    iKnowBase.log ("MySample.onMessage", response);
    if (response.status == 200) { // HTTP status code=200 means normal
response.
        try {
            // First decode the message_format=IKB wrapping as JSON
            var ikbMsg = jQuery.parseJSON(response.responseBody);
        } catch (e) {
            iKnowBase.log('This does not look like a valid JSON:
',response.responseBody);
            return;
        }

        // Check if this is a userList message from the Instant Server
        if(ikbMsg.messageType.indexOf("IKB.USERLIST")>-1){

            if(ikbMsg.messageType.indexOf("IKB.USERLIST.USERS")>-1){
                // USERLIST: Add the users to the displayed user list, if not
already present
                for(var i=0;i<ikbMsg.data.length;i++){
                    if($("#members ul").find("[name='"+ikbMsg.data[i].username
+'']").length==0){
                        $("#members ul").append("<li
name='"+ikbMsg.data[i].username+'>"+ikbMsg.data[i].label
+" ("+ikbMsg.data[i].username+)"</li>");
                    }
                }

            }else if(ikbMsg.messageType.indexOf("IKB.USERLIST.JOIN")>-1){
                // JOIN: Add the user to the displayed user list, if not
already present
                if($("#members ul").find("[name='"+ikbMsg.data.username
+'']").length==0){
                    $("#members ul").append("<li name='"+ikbMsg.data.username
+'>"+ikbMsg.data.label+" ("+ikbMsg.data.username+)"</li>");
                }

            }else if(ikbMsg.messageType.indexOf("IKB.USERLIST.LEAVE")>-1){
                // LEAVE: Remove the user from the displayed user list
                $("#members ul").find("[name='"+ikbMsg.data.username
+'']").remove();
            }
        }else{
            // Normalt user message - display in chat
            $('#messages ul').prepend($('#<li></li>').text("IKB message
received: " +
                                                    "fromUser=" +
((ikbMsg.fromUser)?ikbMsg.fromUser.label:null) +
                                                    ", toUserName=" +
ikbMsg.toUserName +
                                                    ", messageType=" +
ikbMsg.messageType +
                                                    ", data=" +
ikbMsg.data));
        }

    }else{

```

```

        iKnowBase.log('ERROR: Expected HTTP status code 200, but got HTTP
status=' + response.status, response);
    }
}

function subscribe() {
    channel = iKnowBase.Instant.Atmosphere.subscribe({
        topicName: topicName,
        subscriptionOptions: {
            requestUserList: true,           // We want an initial user
list
            subscribeUserListChanges: true // We want to recieve joining/
leave messages
            // Authentication was added in project wide settings, but
could also be added here
        },
        atmosphereOptions: {
            onMessage: onMessage // Use page specific onMessage callback
        }
    });
}

function sendMessage(){
    var message = jQuery("#messageToSend").val(); // Retrieve the message
    iKnowBase.log("sendMessage: " + message);    // Log the message
    channel.push({                               // Send the message to
topic
        toUserName: ''
        , messageType: 'SAMPLE_MESSAGE'
        , data: message
    });
    jQuery("#messageToSend").val("");           // Clear the input
message element
}

window.MySample = {
    subscribe: subscribe,
    sendMessage: sendMessage
}

}());

MySample.subscribe(); // Subscribe to topic

</script>

<div>
    <h4>type your message here:</h4>
    <input id='messageToSend' type='text' /><button id='sendMessage'
name='sendMessage' type='button' onClick='MySample.sendMessage()'>Send
message</button>
</div>

<div id="messages">
    <h4>Topic messages</h4>
    <ul>
    </ul>
</div>

<div id="members">

```

```
<h4>Members</h4>
<ul>
</ul>
</div>
```

Content updates notifications Using Database Integration

This sample shows notifications when changes to documents relevant for the content viewer are available.

Prerequisite: Included project wide settings with authentication (see *Building solutions > Project wide sample with secure token authentication*).

The building blocks for this sample are:

- Page with
 - Page Component: Content viewer: Will retrieve relevant documents
 - Page Component: Template Viewer: Instant publish/subscribe sample code
- Event on the relevant documents
 - Call ikb database function publishing an Instant Message
- Database function: Event procedure: Publish change to Instant Blog
 - Call database procedure PUBLISH_TOPIC1
- DB: Procedure for publishing an Instant Message: PUBLISH_TOPIC1

Instant publish/subscribe sample code

```
// Page-specific application script
(function () {

    var topicName = "/Topic1";           // Using default messageFormat=TEXT
    for this topic

    function onMessage (response) {
        iKnowBase.log ("blog.onMessage", response);
        if (response.status == 200) {
            iKnowBase.notify({
                type:'info',
                title:'Updates for Content Viewer documents are available',
                text:response.responseBody,
                duration:5000 // Remove after 5 seconds
            });
        }
    }

    function subscribe() {
        iKnowBase.Instant.Atmosphere.subscribe({
            topicName: topicName,
            subscriptionOptions: {
                // Not using any custom options
            },
            atmosphereOptions: {
                onMessage: onMessage // Use page specific onMessage callback
            }
        });
    }

    window.MySample = {
        subscribe: subscribe
    }

})();
```

```
MySample.subscribe(); // Subscribe to topic

</script>
```

Event

The Event is set up in /ikbStudio/advanced/events with

- Event for: Documents
- Name: MySampleEvent for Topic1
- Sort key: 1
- Event procedure: Publish change to Topic1
- Document type: Topic1Documents
- Operation: insert, update, delete, enable
- Conditions: none

IKB database function mapping

The database function is set up in /ikbStudio/advanced/dbfunctions with

- Context: Event procedure
- Name: Publish change to Topic1
- PL/SQL command: PUBLISH_TOPIC1

Database procedure PUBLISH_TOPIC1

```
CREATE OR REPLACE
PROCEDURE PUBLISH_TOPIC1
(
  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';
  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
  );
END;
```

Livestream Using Database Integration

This is the same example as “Content updates notifications Using Database Integration”, but instead of just displaying a notification we’ll reload the portlet and display its content.

Warning: Fetching server content based on instant notifications does come with a significant cost. Whenever there event is triggered, all clients currently displaying (and subscribing) to the live feed will immediately request new content. If there are many clients listening, they will all generate requests at the

same moment. This type of live feed is not cached by the iKnowBase application and a high number of clients may overload the system.

First we'll mark the Content Viewer Page Component with markup id = "BlogDocuments". We'll use this ID when reloading.

Second, change the onMessage function from iKnowBase.notify to

```
iKnowBase.PageEngine.reloadComponent("BlogDocuments"); // Reload component by  
markup id on any message
```

Livestream Using Cache Refresh Events

See *Instant notifications of content cache updates* in the *iKnowBase Development Guide > Performance*.

11. Performance

Concept

There are many ways to boost your website's performance. The three main areas that you can work on are: infrastructure, server-side performance, and front-end performance.

- **Infrastructure** includes network, switches and hardware. In general, the faster, the better; either way, there is little iKnowBase or your iKnowBase application can do to help.
- **Front-end performance** is all about the resources that end up on the client, such as the HTML, CSS, JavaScript, and images. iKnowBase does not provide any tools for optimizing this, but instead refers to client tools built for this purpose.
- **Server-side performance** covers the processing spent on producing the content being served to the client.

iKnowBase provides mechanisms to monitor your website and to optimize server-side computation, i.e. the time it takes for the server to produce the content/web page. The chapters below describes those mechanisms.

Monitoring

The iKnowBase java-applications are built with run-time monitoring and visibility into performance metrics. These tools are mostly available under `/ikb$console`, which is documented in *iKnowBase System Administration*.

The documentation is brief, and you may need to click around a bit to familiarize yourself with the tools. Here are a few pointers, though:

- From `/ikb$console`, you may switch on a developer mode which gives quick access to a development toolbar on all page engine pages, as well as per-session logging.
- You control both logging levels and logging targets from the console. There is a RingBuffer/Memory-target which allows for quick access to the last 5000 log entries throughout the system, and there is a Session-based memory target which allows access to the last 10 page requests.
- There are performance probes throughout the system, available under the Monitor-tab.
- There is an `/ikb$runner`-endpoint, which runs a single component and makes the logs for that component readily available.

Caching on the client

Significant performance can be gained by tuning different cache areas. These areas can be divided into two main sections:

1. Static shared resources
2. Application cache

Static shared resources (/ressurs)

Static shared resources deployed to the `/ressurs`-directory can be cached in web cache capable software and the user's browser using Cache Control Directives. Cache Control Directives must be configured on the web server serving `/ressurs`-content or any other capable intermediate server between the web server and the client.

Cache Control Directives are documented in RFC 2616 and must be adjusted according to the customer's needs, but the following directives are typically applicable as response directives:

- `max-age=<number of seconds>` (see RFC 2616, chapter 14.9.3)
- `public` (see RFC 2616, chapter 14.9.1)
- `must-revalidate` (see RFC 2616, chapter 14.9.4)

Apache Web Server can be configured as follows:

```
#Cache all /ressurs-files for 15 minutes in shared cache for all users.
Revalidate after 15 minutes is mandatory
<LocationMatch "(?i)^/ressurs/.*$">
  Header set Cache-Control "max-age=900, public, must-revalidate"
</LocationMatch>
```

Caching content from the content server (/Content)

HTTP Response cache directives to client based on client input

When accessing contents provided by ikbViewer Content Server, certain URL-options can be used to control the Cache Control Directives returned to the client. For supported options, see *iKnowBase API Reference > Content Server > Download content*.

An example where cache directives with URL-options is applicable, is when using images stored in the iKnowBase content store as part of a relatively static web layout. The image URL /Content/40028/SampleImage.jpg will not return any cache headers (unless intermediate servers are configured to override). To cache the image for 60 seconds, use /Content/40028/expires=60/SampleImage.jpg. This will return HTTP Response Header "Expires: <now + 60 seconds>".

HTTP Response cache directives to client for autogenerated URLs to iKnowBase content

The ContentViewer will sometimes generate URLs to content, such as images or documents stored in document attributes. Whenever the ContentViewer is capable of generating a URL with proper caching directives without loading more data from the database, it will.

Caching on the application server

The iKnowBase application can cache data in memory, in order to avoid much more expensive operations involving the database.

Metadata Cache

Metadata for the components used in iKnowBase will be automatically cached in memory by the server. The metadata cache is revalidated against the iKnowBase repository if more than 10 seconds has passed between last revalidation and the current request. Data is reloaded from iKnowBase repository if any change is detected.

Metadata cache is by default replicated to other application instances using the same database, schema and network multicast group.

An administrator can force a cache clearing by using /ikb\$console.

Page engine content cache

In iKnowBase you can enable content caching for any portlet on any page. When content caching is enabled for a portlet, the server will store the result of the first execution of the portlet in memory, hereafter called the cache. When further requests are made to the same portlet, the server will go to the cache, avoiding the overhead incurred by producing the result over again.

Content caching is suitable for portlets which displays information that seldom changes or information that is not critical to present in "real-time". You need a deep understanding of the usage patterns in order to configure content caching appropriate. Beware that the content cache consumes memory. But, it also frees up resources by reducing the number of request to the server and server processing.

An administrator can force a cache clearing by using ikb\$console or restarting the application/jvm.

Configure content caching

You configure caching per page component (a portlet living on a page). You may choose between four predefined content cache strategies as well as add your own definitions using Development Studio:

- **Shared, 1 hour:** The content of the portlet will be cached per domain and language; all users with the same language on the same domain will share the same cached content. The content will expire one hour after the first access.

- **Shared, until next date:** The content will be cached per domain and language; all users with the same language on the same domain will share the same cached content. The content will expire after midnight (server time).
- **Per user, 1 hour:** The content of the portlet will be cached per user, domain, and language. The content will expire one hour after the first access.
- **Per user, until next date:** The content of the portlet will be cached per user, domain, and language. The content will expire after midnight (server time).

When a portlet does not have content caching, it will most often be rendered with a *portlet container*, a set of div-tags surrounding the portlet; for some portlets (such as the content viewer), this can be specified as part of the portlet definition. For cached portlets, however, the decision whether to render a portlet container belongs to the page template. You **must** use the freemarker template RegionModel macro directive to specify whether to render a portlet container:

```
[@page.regions region="north" /]
[@page.regions region="north" container="true" /]
[@page.regions region="north" container="false" /]
```

Access control

Note that for a cached portlet, access control is applied when the content is first generated, and not when it is rendered. For example, if a portlet is set up with a shared content cache, the first user to access the portlet (before the content cache is populated) will generate the content, and subsequent users will see that exact content. Hence, if the portlet would generate more content to the first user than the next ones, due to security settings, the subsequent users would see more content than they should. You should therefor **only** use the shared cache when the content is public, or when all users who may access the portlet has access to the same data.

Refreshing the content cache

The content cache will be updated whenever a client requests the content of a cached portlet, and the time-to-live of the already cached content has expired. There is no way to have the server automatically detect and refresh the content outside of a client request.

Instant notifications of content cache updates

iKnowBase comes with client and server side support for creating applications with real time asynchronous messaging support, commonly referred to as iKnowBase Instant (see the chapter on this component for more information).

The iKnowBase PageEngine content cache has pre-built integration with iKnowBase Instant, covering the following basic areas:

- Publishing update notifications to iKnowBase Instant whenever content is regenerated.
- Rendering information that identifies the various cached portlets on a client page
- Using client side javascripts to listen for update notifications, reloading portlets that have been updated

The basic setup requirements are as follows:

- In Development Studio, you must create a "page cache strategy" with the flag "publish updates to instant"
- In your page, the relevant page components must be configured to used the new page cache strategy
- In your page template, the region in which the page components is rendered must be configured to render a portlet container
- Also in your page template, you must start the client script, ref the *iKnowBase API Reference* (but typically `iKnowBase.PageEngine.InstantContentCache.start();`)

Note that using instant notifications does come with a significant cost. Whenever there is a cache update of a portlet, all clients currently displaying that portlet (with the same cache key) will immediately request new content. If there are many clients listening, they will all generate requests at the same moment. While the cache infrastructure is quite fast, if the number of clients get too big, this may overload the system. This problem is severe enough that cache theory has given it it's own name, "thundering herd".

Setup

The content cache mechanism needs to be enabled for the site in order to use this functionality. By default, there is one shared content cache per database schema. This content cache is replicated between application servers running on the same subnet, using the same database schema. See the *iKnowBase Installation Guide* for further information.

Clustering and the application server cache

The application cache (JVM) will by default replicate to other application instances if they

- Share the same database name and schema (user)
- Support multicast network traffic (between the instances/servers)
- Configured with the same multicast group

A default multicast group is configured in the application, but can be changed by an administrator. See *iKnowBase Installation Guide > Configuring the ikbViewer application > CacheManagerConfiguration* for options.

Render Strategy

In iKnowBase you must specify a render strategy for a portlet on a page. The render strategy decides when to produce and render the portlet. This is a simple mechanism which enable you to delay the production and rendering of certain parts of a page, which enable quick rendering of the most critical parts of a page.

You can choose between three different render strategies:

- **Inline:** The portlet will be loaded and rendered by the browser together with the page.
- **On load:** The portlet will be loaded and rendered after the page is rendered by the browser.
- **Manual:** The portlet will not be automatically loaded and rendered by the browser. The developer must implement code to render the portlet using the JavaScript function `reloadComponent`.

Note: If you specify other render strategies than **Inline**, portlet decorations will automatically be rendered for the portlet (i.e. a div-tag with id and class attributes).

The iKnowBase component reload mechanism requires that your pages are served by the iKnowBase page engine, and that the component being refresh was in fact served by a page available at the URL shown in the client browser. That means that these two scenarios will not work:

- You cannot use **On load** for a page which is manually loaded using ajax. For example, if page “a” does a manual ajax call to load page “b”, **On load** will not work for any component served by page “b”.

12. Integrating with an LDAP Directory

Many organizations have established a central repository (a directory service) for user and group information, making it possible to edit that information independently from all the different systems that need it. However, iKnowBase needs a local copy of that information in order to support security and context operations.

Use the Advanced LDAP Synchronization (formerly known as Advanced OID Synchronization) component in iKnowBase Development Studio to set up a replication link, where information automatically flows into iKnowBase when the directory service changes it.

Understanding LDAP synchronization

During user synchronization, the synchronization engine will periodically read the LDAP changelog to understand what kind of changes have happened in the source user directory.

The changelog will contain entries indicating that users, groups or group memberships have changed. The synchronization engine will use its configuration to copy these changes into the corresponding iKnowBase objects.

- For more information on synchronizing a person, see [Synchronize a Person](#).
- For more information on synchronizing an organization, see [Synchronize an Organization](#).
- For more information on pre user synchronization and post user synchronization, see [Pre and Post User Synchronization](#).
- For more information on optional attributes, see [Optional Attributes](#).

Synchronizing users

This chapter describes the options that control what happens when the LDAP changelog contain a user object. The options include

- Create user objects, such as an iKnowBase user, a user dimension and a person card
- Create organization objects, such as an iKnowBase group and an organization card
- Define optional attribute mappings for the person and organization cards
- Specify PL/SQL procedures to run at defined points in the process.

User objects

When the changelog specifies changes to a person in the LDAP directory, the user synchronization engine will create or update iKnowBase information. The creation and updates to this information is managed by information from the synchronization profile:

- Every user has a user object in the iKnowBase user table:
 - On tab two, “person card mapping” specify which LDAP-attributes contain username, e-mail and display for the user object itself. You must enter valid values corresponding to your environment; typical values can be “UID”, “MAIL” and “displayName” respectively.
 - On tab one, “default ikb group” specifies an iKnowBase group that the user will e added to.
- Further, every person has a personal dimension, an iKnowBase information object for that particular person:
 - On tab one, “Top dimension (person)” defines the iKnowBase dimension under which the personal dimension is created
 - On tab one, “Dimension type” defines the type of dimension which is createdgr
- Further, every person has a “person card”, an iKnowBase information object describing the user. The creation of this card is managed by other information from the synchronization profile
 - On tab one, “Document type (person card)” defines the iKnowBase information object type used when creating the person card.
 - On tab one, “additional attributes” specify a list of LDAP-attributes that will be added to the person card during creation or update

- On tab one, “default access group” specifies the ACL used for the person card.

Organization objects

A user entry in LDAP typically contains some sort of information that identifies the organization unit that the user belongs to, such as his department or section.

The iKnowBase LDAP Synchronization engine is capable of automatically creating iKnowBase object corresponding to the organization. Check the relevant checkboxes on tab two to specify these:

- You may create an iKnowBase group, with all users with the same organization attribute as members;
 - On tab two, check the box "create ikb group"
- You may create a “organization card” for the organization, and iKnowBase information object describing the organization:
 - On tab two, check the box "create org. card"

Optional Attributes

The default synchronization will add only key information such as name and external key to the user and organization cards. On tab two, you may choose to add extra attributes to both of these cards, with values from available attributes of the LDAP user object.

Pre and Post Plug-Ins

In the case of users, there can be issues related to synchronization that are not resolved in the configuration interface. For example, all the users in an organization have an e-mail address in the employeenumber@company.com format. In this case, customer cannot use the standard configuration interface to map the employee number to an iKnowBase attribute.

You can specify the functions that run the pre user synchronization and post user synchronization to iKnowBase. These functions take some parameters to make changes to the objects that are related to the user synchronization.

The interface provides a plug-in in which the developer can write a code and use this code as a plug-in to the synchronization application. The synchronization application supports two plug-ins, the preUserSynch and postUserSynch plug-ins.

Pre user synch plug-in

The preUserSynch plug-in is called before the synchronization takes place in iKnowBase. The postUserSynch plug-in is called after the actual synchronization takes place in iKnowBase.

The signature of the plug-in is:

```
CREATE OR REPLACE PROCEDURE preuseradd (
  p_profile_name    IN      VARCHAR2,
  p_dn              IN      VARCHAR2,
  p_ldap_conn       IN      DBMS_LDAP.SESSION,
  p_changetype      IN      VARCHAR2,
  p_entrytype       IN      VARCHAR2, (PERSON, GROUP, DELETE)
  p_cn              IN      VARCHAR2,
  p_continue        OUT     NUMBER
  p_do_create_card  OUT     NUMBER
)
```

Parameter	Description
p_profile_name	This variable contains the name of the synchronization profile that calls this plug-in.
p_dn	This variable contains the distinguished name (dn) string of the object that is synchronized.
p_ldap_conn	This variable contains the LDAP connection established with the remote user directory. This

	variable can be used to operate directly in the connected user directory or as a variable for public APIs exposed by IKB_OID_SYNC.
p_entrytype	This variable contains the entry type. Values are PERSON, GROUP or DELETE.
p_cn	This variable contains the cn-value.
p_ldap_conn	This variable contains the LDAP connection established with the remote user directory. This variable can be used to operate directly in the connected user directory or as a variable for public APIs exposed by IKB_OID_SYNC.
p_continue	This variable is a numeric value that indicates whether to continue the synchronization of the object or not. The value 1 indicates that the synchronization must continue and value 0 indicates that the synchronization must stop. The developer can extend the functionality of the application using a programming code. This programming code can manipulate the object that is synchronized, and decide if the application must continue synchronizing the object or not.
p_do_create_card	This variable is a numeric value that indicates whether to create person card or not. The value 1 indicates that the person card will be created, and value 0 indicates that the it will not.

Post user synch plug-in

The postUserSynch plug-in is called after the user, the organization card, and the person card is created in iKnowBase.

The signature of the plug-in is:

```
CREATE OR REPLACE PROCEDURE postuseradd (
  p_profile_name    IN   VARCHAR2,
  p_dn              IN   VARCHAR2 DEFAULT NULL,
  p_ldap_conn      IN   DBMS_LDAP.SESSION DEFAULT NULL,
  p_user_id        IN   NUMBER DEFAULT NULL,
  p_user_doc_id    IN   NUMBER DEFAULT NULL,
  p_org_doc_id     IN   NUMBER DEFAULT NULL,
  p_changetype     IN   VARCHAR2,
)
```

Parameter	Description
p_profile_name	This variable contains the name of the synchronization profile that calls this plug-in.
p_dn	This variable contains the distinguished name (dn) string of the object that is synchronized.
p_ldap_conn	This variable contains the LDAP connection established with the remote user directory. This variable can be used to operate directly in the connected user directory or as a variable for public APIs exposed by IKB_OID_SYNC.
p_user_id	This variable contains the internal iKnowBase ID of the actual user that is created. If the user is not created, the value of this variable is NULL.
p_user_doc_id	This variable contains the ID of the personal card that is created for the user. If the personal card of

	the user is not created, the value of this variable is NULL.
p_org_doc_id	This variable contains the ID of the organization card that is created for the user. If the organization card of the user is not created, the value of this variable is NULL.

Synchronizing groups

This chapter describes the options that control what happens when the LDAP changelog contain a group object. The options include

- Map all LDAP groups to iKnowBase groups
- Map only selected LDAP groups to iKnowBase groups

Decide which groups to map

On tab one, the checkbox “sync all groups” define what happens during sync:

- When the box is checked, all groups from the LDAP-source are automatically synchronized with the corresponding iKnowBase group
- When the box is not checked, only groups that already exist in iKnowBase will be synchronized. The relationship between the two objects are controlled by the external_key of the group, which will be of the form “IKB_OID_SYNC:IKB_ADMINISTRATORS” for a group named “IKB_ADMINISTRATORS”

Hierarchical groups

The LDAP user directory typically supports groups inside groups, to easily model hierarchical relationships. iKnowBase does not support the same relationships, and hence will need to flatten the groups.

For example, in LDAP, the group “all” may contain the groups “sales” and “service”, which in turn contain all the users. This group “all” in iKnowBase will then contain all the users directly. The functionality will be the same; in fact, the flattened structure will be faster than using a hierarchical lookup.

Scheduling a Synchronization

Synchronization can run in a batch or a command line utility (such as Sql*Plus). Two methods use the same procedure to run the synchronization.

The procedure has the following signature:

```
oid_sync.synchronize_users (
p_sync_id      IN NUMBER,
p_dn           IN VARCHAR2 DEFAULT NULL,
p_changetype  IN VARCHAR2 DEFAULT NULL,
p_bootstrap   IN BOOLEAN DEFAULT NULL,
p_debug       IN BOOLEAN DEFAULT FALSE,
p_dbmsoutput_debug IN BOOLEAN DEFAULT FALSE);
```

This is the main procedure and the gate to start the synchronization. With only the mandatory parameters supplied, the synchronization uses the LDAP changelog to find which objects must be synchronized.

Parameter	Description
p_sync_id	This variable contains the current synchronization profile ID.
p_dn	This variable contains the dn that the developer wants to synchronize. This variable must be specified with the p_changetype variable.

p_changetype	This variable contains the description of the LDAP operation. The values can be Add, Modify, or Delete.
p_bootstrap	This variable is a boolean variable. If the developer wants to synchronize all users from the LDAP to iKnowBase, this variable is set to TRUE.
p_debug	If the variable is set to TRUE, this variable creates log entries in the OID_SYNC_ERRORLOG log table.
p_dbmsoutput	If the variable is set to TRUE, this debug creates log information to terminal. This is applicable only when the procedure starts from a command line utility (such as Sql*Plus). Before you run the procedure, ensure that you run: set serveroutput on size 1000000.

LDAP provider specific requirements – OpenDJ

Incremental user synchronization requires access to changeLog entries. A default installation of a **single** (i.e. standalone) OpenDJ directory server will not enable the “External Change Log”. (A replicated setup with more than one server will enable “External Change Log”.) See OpenDJ documentation for details regarding enabling the “External Change Log”.

Tips

- <http://ludopoitou.com/2011/05/11/opensdj-enabling-the-external-change-log-on-a-single-server/>
- <https://onemoretech.wordpress.com/2012/05/03/enabling-changelog-on-a-standalone-opensdj-server/>

13. Integrating with Oracle Secure Enterprise Search

The iKnowBase crawler for Oracle Secure Enterprise Search (SES) is an optional component that makes it possible to index the iKnowBase content through the premier standalone search engine of Oracle.

The iKnowBase crawler supports the following features:

- Deliver any content in the iKnowBase Content Store to Oracle SES for indexing.
- Control which documents and attributes are indexed.
- Security with the use of existing ACL-mechanisms in iKnowBase.
- Display of documents in context through the use of Target selectors in iKnowBase.

Use the Crawler definition component in iKnowBase Development Studio to set up an iKnowBase crawler.

Concept

The following are the key concepts around which the iKnowBase crawler for Oracle Secure Enterprise Search is built.

To index data:

- In iKnowBase, you can use a Search source to control which documents and attributes that are indexed.
- In iKnowBase, you can use a Crawler definition to control the state of indexing. The Crawler definition refers to a search source, and adds state information that lets Oracle SES index only new and changed documents.
- In Oracle SES, you can create and run an iKnowBase source to index the data. A source is configured to use a crawler definition in iKnowBase. As part of indexing the data, Oracle SES stores information on which access control list (ACL) is applicable to each document.

Note: You can use different sources to facilitate SES searches in subsets of the available information in iKnowBase.

To search for data:

- In Oracle SES, you can execute a query. Oracle SES asks iKnowBase for a list of ACLs that are accessible to you as a user. Oracle SES caches this list, to reduce the load on the repository.
- The search results that the Oracle SES returns refer to a redirector servlet in iKnowBase. This servlet verifies document access, and decides the context to display the document. This servlet also utilizes the target selector mechanism in iKnowBase to decide the actual page.

Note: Configuration of target selectors is a critical task in using Oracle SES to search for and present the iKnowBase content.

Install the iKnowBase adapter into Oracle SES

To set up an iKnowBase source in SES, perform the following steps:

1. Install the iKnowBase crawler adapter (iKnowBase-7.1.7-SESCrawler.jar) into \$ORACLE_HOME/seshome/search/lib/plugins/iknowbase/, and restart Oracle SES.
2. From the Oracle SES Administration Console, under "global settings", create a new "source type" with these parameters:
 - Set the name to "iKnowBase"
 - Set the description to "iKnowBase Crawler for Oracle SES"
 - Set the plug-in manager class name to "com.iknowbase.search.oracleses.IKBCrawlerPluginManager"
 - Set the plug-in jar file name to "iknowbase/iKnowBase-7.1.7-SESCrawler.jar"

Create a source

To set up an iKnowBase source in SES, perform the following steps:

1. From the iKnowBase Development Studio, define a Search source, selecting the documents and attributes you want to index. Note: The names given to the attributes are set as the search attributes in Oracle SES.
2. From the iKnowBase Development Studio, create a Crawler definition.
 - Specify the Search source you want to use, defining the source data to be crawled.
 - Select the Domain for which to index data. This is used during a search, when the Target selector decides in which context documents should be opened. Note: This crawler definition id is used when defining the iKnowBase source in Oracle SES.
3. In iKnowBase, make sure that there are Target selectors for all relevant documents.
4. From the Oracle SES Administration Console, define a source for the iKnowBase crawler
 - Specify the database connection information (database URL, username and password) to connect to the database.
 - Specify the iKnowBase Crawler definition id, which identifies the crawler definition in iKnowBase.
 - “Delete index on crawl” makes the crawler recreate the index on each crawl. This means that the index is deleted at the beginning of each crawl, and that all documents must be fetched for each crawl. This is not a sustainable idea for production use, but very convenient when setting up and testing a new search source.
 - In the Authorization page, leave defaults enabled.
5. From the iKnowBase Development Studio, after creating the iKnowBase crawler in Oracle SES, edit the crawler definition. Under “crawling parameters”, set the property “number of threads” to 1.

Now you have created an iKnowBase Crawler in Oracle SES.

Schedule a source for indexing

After you have created an iKnowBase source, you are ready to start crawling.

To conduct a test crawl and enable automatic scheduling, perform the following steps:

1. On the Oracle SES Administration Console, define a SES crawl schedule.
2. Run the schedule manually, to verify the crawler log.
3. If the crawler log is accurate, enable automatic scheduling.

Security

The iKnowBase crawler supports a secure search based on the Oracle SES ACL-mechanism.

- During a crawl, (ie indexing of data), each document is stamped with its iKnowBase ACL.
- While searching for a document, Oracle SES asks iKnowBase for a list of ACLs that the logged in user is a part of. This list is applied to the query, which returns only the documents with the available ACLs.
- When you click a document link to open it, the regular iKnowBase security mechanisms are used.

The security mechanism has the following limitations:

- Oracle SES stores the ACL on each crawl. If the ACL of a document changes, this change is propagated to the Oracle SES index on the next crawl. Until the next crawl, the original ACL is applied to all search operations.
- Oracle SES caches the list of ACLs for a given user for a specific period of time. During this time, Oracle SES does not apply changes to the ACLs that are available for the user. How long this list is cached, can be configured in the Oracle SES Admin application.
- Oracle SES applies ACLs for a user to each search operation. The number of ACLs per user should be limited, due to performance. The threshold depends on multiple factors, mostly related to the hardware on which Oracle SES is running.

Search User Interface

The iKnowBase crawler is not related to the user interface that is used to search the data. You can use any available mechanism, such as the standalone GUI, predefined portlets, or custom code.

Custom filter plugin

The iKnowBase crawler exposes a plug-in extension point, where it is possible to add a filter which is used to filter documents before indexing.

No default plug-in is provided. Perform the following to implement and configure a plug-in:

- Implement the interface `oracle.search.sdk.crawler.DocumentServiceManager`.
- Integrate the plug-in into the same jar-file as the iKnowBase crawler (crawler plugin), to avoid class loading issues.
- From the Oracle SES Administration Console, enter the plug-in class name on the parameter page of the iKnowBase crawler (crawler plug-in).

The plug-in will perform much the same job as any filter in a document pipeline. However, since this plug-in is executed before Oracle SES can see the data, it is also possible to replace the document content itself inside this filter.

14. Integrating with an E-mail Server

iKnowBase E-mail Reader is a stand-alone component that enables you to automatically load mails from a mailbox into the iKnowBase Content Store. It has the following key features:

- Reads e-mails from any POP3-compliant mail server. Enables you to configure e-mail accounts declaratively using the E-mail integration component in iKnowBase Development Studio. It is possible to define multiple e-mail accounts.
- Loads e-mail messages including recipients, body parts, and attachments into a set of database tables, ready for further processing.
- Calls a PL/SQL procedure (message handler) to process the e-mail messages. By default, a procedure exists. However, you can also create your own procedure with different properties.
- Enables you to schedule and run it on the defined e-mail accounts once or at defined intervals.
- Enables you to run it as a service under Windows, Linux, and various Unix platforms using third-party components such as Java Service Wrapper (<http://wrapper.tanukisoftware.org/doc/english/download.jsp>).

Configuring E-mail Accounts

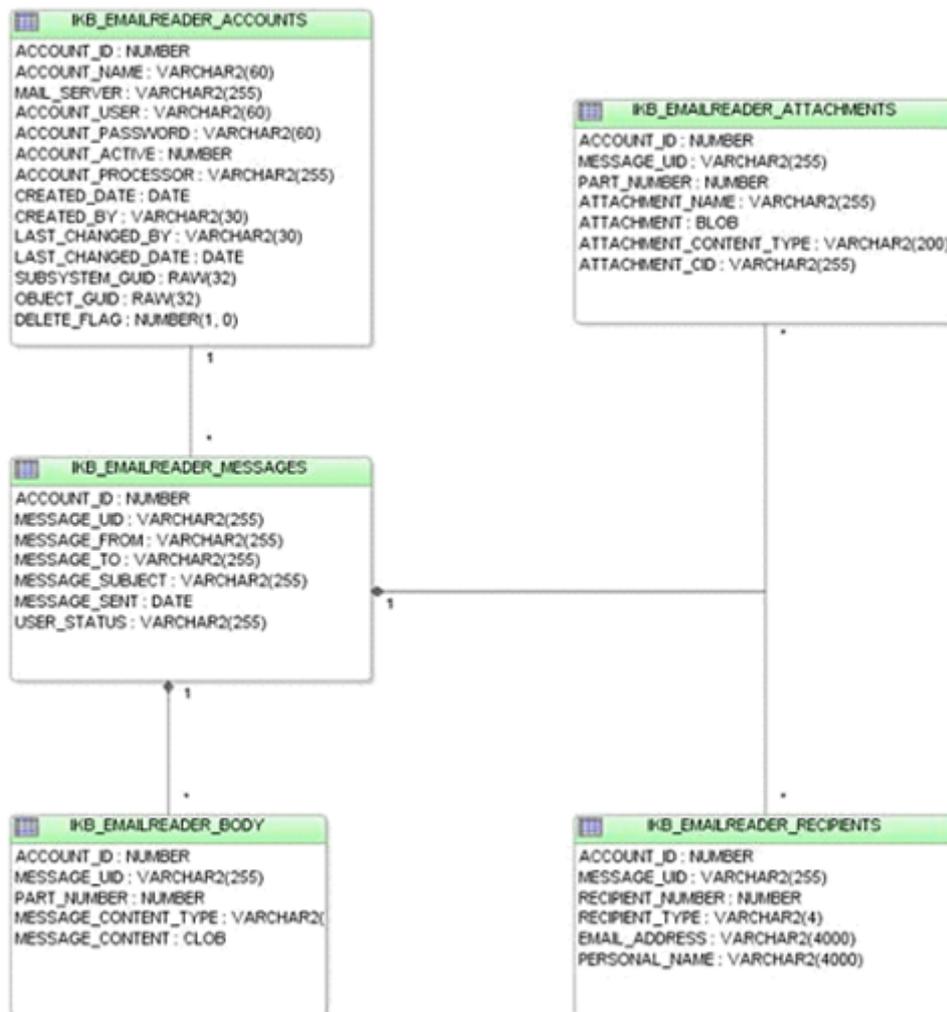
Use the E-mail integration component in iKnowBase Development Studio to configure e-mail accounts used by the iKnowBase E-mail Reader. For more information on configuring an e-mail account, see the iKnowBase Development Reference.

Process Flow of the E-mail Reader

The iKnowBase E-mail Reader program performs the following tasks, once for each active account during execution:

1. It connects to the mail server using the assigned credentials. The connection is made using the POP3-protocol, supported by most mail servers.
2. It loads a list of e-mails from the inbox of the specified account and processes each message.
 - If the message exists in the E-mail Reader tables in the database, the message is skipped.
 - If the message doesn't exist in the E-mail Reader tables in the database, the message is downloaded, split into parts and stored in the database.
 - If the account is set up with Delete messages after copying, the message is marked for deletion on the mail server.
3. After copying all messages, the connection to the mail server is closed with a confirmation message with the flag Permanently delete all messages marked for deletion. If the flag is checked, all the messages will be deleted from the mail server.
4. The message handler is called to process the new messages, typically to load them into the iKnowBase Content Store.

E-mail Reader data model



There can be multiple body parts of a single e-mail. This happens because of the e-mail programs that send multiple versions of the body of an e-mail. One version can be a rich HTML-message and the other can be a simple text-message.

To select the body part that you want to use, use the following rules:

- If you want to select a HTML body part, search for the content_type text/html in the body of the e-mail.
- If you want to select a pure text body part, search for the content_type text/plain in the body of an e-mail.
- Other content-types are specific to individual clients, and have priority after earlier mentioned rules.

Message Handler

The E-mail integration accounts can be configured with a message handler (e-mail reader procedure) that is a customizable PL/SQL procedure. You can use the message handler for automatically loading of e-mails from the iKnowBase E-mail Reader tables into the iKnowBase Content Store. Loading e-mails into the iKnowBase Content Store can be useful for various reasons and the actual document structure required varies widely.

When the iKnowBase E-mail Reader program runs, it populates the iKnowBase E-mail Reader tables with e-mail information. After populating these tables, the program activates the message handler, passing the account_id of the account as the only parameter.

The message handler will typically traverse the iKnowBase E-mail Reader tables, using the iKnowBase Service API to create the required iKnowBase documents and objects.

iKnowBase comes with a default message handler TEST_INSERT_MAIL, useful for testing and simple setups. This message handler has the following features:

- Runs through all e-mails where the IKB_EMAILREADER_MESSAGES.USER_STATUS is blank.
- Creates an e-mail document for every new e-mail. It is set up with a few external codes to identify the document type and the attributes for both the sender and receiver.
- For each e-mail message, the message handler runs through all attachments and creates them as sub documents to the main e-mail document.
- If the e-mails are saved without errors, the IKB_EMAILREADER_MESSAGES.USER_STATUS is set to LOADED. If some error occurs while saving, an error message from `ikb_integration.integrate_objects` appears.

Reliable Messaging

The iKnowBase E-mail Reader guarantees once-and-only-once semantics of message delivery and processing, given that the message handler is built according to certain principles, see the last bullet point of this section.

- The iKnowBase E-mail Reader retrieves all messages from the mail server.
- No messages are deleted from the server until they have been committed to the database in a transaction.
- The iKnowBase E-mail Reader does not duplicate a message in the database (using the MESSAGE_UID from the mail server to identify the message).
- If messages are deleted from the iKnowBase E-mail Reader tables, but still exist in the mail server, a problem occurs. This happens if the mail server fails during message deletion. To avoid this problem, use the field IKB_EMAILREADER_MESSAGES.USER_STATUS to indicate that a message has been processed, rather than deleting the message. The iKnowBase E-mail Reader program can then detect duplication even after the message has been processed.

Running the iKnowBase E-mail Reader

The E-mail Reader program is distributed as a java .jar-archive, which can be run using the following command:

```
java -jar iKnowBase-<version>-EmailReader.jar
```

where version is the actual version of iKnowBase Content Services.

The E-mail Reader program accepts the following parameters.

Parameter	Description
-url <url>	JDBC-url to database
-username <user>	Database username, for iKnowBase user/schema
-password <pwd>	Database password, for iKnowBase user/schema
-interval <secs>	This is the interval between runs. The program runs repeatedly with an interval of the specified number of seconds. If this parameter is missing, the program runs only once. The first run occurs immediately.
-properties <file>	Refers to a property file with parameters. In some operating systems where the command line parameters can be seen, this parameter is useful in hiding the username or password from other logged on users.
-log4j <file>	Refers to a custom log4j.properties-file. E-mail Reader program uses a default setup, unless specified.

In general, the recommended setup is to use a property file, and enter all the parameters in that file. The syntax of the file has multiple lines, where each line contains a parameter name, an equals sign, and a value. Lines beginning with a hash sign are comments:

```
# Database connection
url=url
username=user
password=pws
# 3600 seconds = one hour
interval=3600
```

Configuring logging

The iKnowBase E-mail Reader relies on log4j from the Apache logging project for logging. You can configure the log using a custom log4j properties file.

If you do not configure the log, the program uses a default configuration which logs detailed messages to standard out, and warning messages to the ERROR_TAB table in the iKnowBase database. The actual setup for this logging can be seen in the file log4j.properties which is distributed along with the iKnowBase E-mail Reader program.

To customize the logging, create your own log4j properties file with the parameters that you want and then start the iKnowBase E-mail Reader program with the extra `-log4j <filename>` command line argument.

For information on log4j configuration, see <http://logging.apache.org/log4j/1.2/index.html>.

Note: If you want to log to the Windows event log, you need to install the DLL available through the log4j project.

15. Integration Using External Attribute Types

In general, value list values are statically defined in the iKnowBase metadata repository. However, it is sometimes useful to have entirely dynamic value lists. For this, iKnowBase supports attribute types of type EXTERNAL. You can configure and maintain these attribute types in the Metadata Management tool available in the Development Studio.

External attribute types enable you to tag iKnowBase information objects with metadata which are defined outside of iKnowBase, e.g. in another system. These attributes can be used as ordinary iKnowBase attributes in Viewers, Forms and Search Sources. They require an external source, which contains functions to support lookups and value lists.

For an external attribute type you must define which external source API to use. You must implement this API to contain given functions for retrieval of information from the external data source.

You can add the external attribute type for a Form component, to enable the end user to tag iKnowBase information objects with this attribute. On the Value tab of the Edit pane for the Form component, you can specify which value list to use for the attribute as well as the default value. The API function `get_list_functions` is used to populate the dropdown box for value lists, and the function `get_values` is used to populate the list of possible default values.

You can add the external attribute type to a Viewer or a Search source, to enable the end user to view and search for information objects tagged with this metadata.

External Source API

Before you can create an external attribute type you must implement the external source API for the external data source. The API is implemented as a plsql-package with the following signature:

```
package my_external_source IS

    function get_value (p_value_id in varchar2) return varchar2;

    function get_values return external_api_table;

    function get_list_functions return external_api_table

    function get_changes (
        p_from_date date,
        p_to_date in date
    ) return external_api_table;

    procedure delete_external_attribute;

end;

table(ikbPropertyTable).
|_. Method          |_. Description|
|get_value          |Retrieves the name for a given item.|
|get_values         |Retrieves a set of values (id and name). You can create several similar functions if you need subsets of the values.|
|get_list_functions |Returns all get_values functions. The list is used when the end user selects between subsets.|
|get_changes        |Returns all changed values (id and name) within the given period. This function is used by the nightly job that re-indexes the iknowbase documents, to update the labels of an external attribute, which may have changed since the last indexing.|
```

|delete_external_attribute |Used for deletion of metadata which no longer exist in the external data source. You must run this function from a trigger or a batch job defined at the external data source. |

Example: Topic Map API

```
CREATE OR REPLACE PACKAGE topic_map_api

/**
 * Example package used by a external attribute. The functions
 * get_value, get_changes and get_list_functions have to be included
 * the package; otherwise external attributes will not work properly.
 */

AS
  /**
   * Get the description/name for a given external attribute.
   */
  FUNCTION get_value (p_value_id IN VARCHAR2)
    RETURN VARCHAR2;

  /**
   * Gets a set of values (id and name). You can create several
   * similar functions if you need subsets of the values.
   */
  FUNCTION get_values
    RETURN external_api_table;

  /**
   * Returns all 'get_values' functions. The list is used when
   * the user selects between subsets.
   */
  FUNCTION get_list_functions
    RETURN external_api_table;

  /**
   * Returns all 'changed' values to the nightly job that reindexes
   * the XML_DATA-field. The purpose is to update the labels in XML_DATA
   * where the label has changed in the external system.
   */

  FUNCTION get_changes (p_from_date IN DATE DEFAULT NULL, p_to_date IN DATE)
    RETURN external_api_table;

  /**
   * Internal function used when external attribute is deleted in
   * the source system. Returns OK. <message> or ERROR:<error msg>
   */
  Function Delete_External_Attribute (p_external_value in varchar2) return
  varchar2;

END topic_map_api;
/

CREATE OR REPLACE PACKAGE BODY topic_map_api
AS

  /**
   * Custom implementation of get_value.
   * Returns the label/description of an external attribute value.

```

```

*/
FUNCTION get_value (p_value_id IN VARCHAR2)
    RETURN VARCHAR2
IS
    l_return_value    VARCHAR2 (200);
BEGIN
    SELECT basename
        INTO l_return_value
        FROM ikbtm.ikb_tm_topic
        WHERE ID = TO_NUMBER (p_value_id);

    RETURN l_return_value;
EXCEPTION
    WHEN NO_DATA_FOUND
    THEN
        RETURN NULL;
END;

/**
 * Custom implementation of get_values. Returns all or a
 * subset of values from the external source.
 */
FUNCTION get_values
    RETURN external_api_table
IS
    CURSOR c1
    IS
        SELECT    external_api_rec (ID, basename)
            FROM ikbtm.ikb_tm_topic
            ORDER BY basename;

    l_values    external_api_table := external_api_table ();
BEGIN
    OPEN c1;

    FETCH c1
    BULK COLLECT INTO l_values;

    CLOSE c1;

    RETURN l_values;
END;

/**
 * Custom impelemtation of get_changes. Returns all or a
 * subset of values that have changed since the last time
 * the indexing job ran.
 */
FUNCTION get_changes (
    p_from_date IN DATE DEFAULT NULL,
    p_to_date IN DATE
)
    RETURN external_api_table
IS
    CURSOR c1
    IS
        SELECT    external_api_rec (ID, basename)
            FROM ikbtm.ikb_tm_topic
            ORDER BY basename;

    l_values    external_api_table := external_api_table ();

```

```

BEGIN
    OPEN c1;

    FETCH c1
    BULK COLLECT INTO l_values;

    CLOSE c1;

    RETURN l_values;
END;

/**
 * Custom implementation of Delete_External_Attribute
 */
FUNCTION Delete_External_Attribute (
    p_external_value in varchar2
)
    RETURN varchar2
IS
BEGIN
    RETURN ikb_pck_document.DeleteExternalAttribute(
        p_site_id => 10,
        p_api_name => 'ikb.topic_map_api',
        p_external_value => p_external_value
    );
END;

/**
 * Custom impelementation of get_list_function. Returns all instances
 * of get_values. Each subset must be represented with its own
 * get_values-function.
 */
FUNCTION get_list_functions
    RETURN external_api_table
IS
    l_list_functions external_api_table := external_api_table ();
BEGIN
    l_list_functions.EXTEND;
    l_list_functions (1) :=
        external_api_rec ('get_values', 'Hent alle verdier');

    /* If more than one, just add like this
    l_list_functions.EXTEND;
    l_list_functions (2) :=
        external_api_rec ('get_values2',
            'Hent alle verdier for topic type XX'
        );
    */

    RETURN l_list_functions;
END;
END topic_map_api;
/

```

16. iKnowBase Service API

The iKnowBase Service API exposes functionality to access and process documents and metadata in the iKnowBase Content Store. The API is available both as functions and procedures in PL/SQL packages, Java methods, and Web Services. This chapter, together with the documentation created from the PL/SQL and Java code, composes the documentation of the API.

Security considerations

When using the iKnowBase Service API, the programmer who implements the api client is responsible for authentication the end user. The programmer will then pass that user identity to the Service API using the parameter `authenticatedUser`, which is present on all methods. The API will then use the security privileges of that user when determining which functionality and data which is available.

API documentation

- For the PLSQL-documentation, see </ressurs/iknowbase/doc/api/iknowbase-database-api/index.html>
- For the Java API documentation, see </ressurs/iknowbase/doc/api/iknowbase-contentservices-api/index.html>
- For the WebServices API, the WSDL is available at the configured web services endpoint, e.g. </ws/iknowbase-2.wsdl>

Sample code

This chapter contains both Java and PL/SQL sample code.

The Java samples uses a variable `ds` which references a `DocumentService` instance.

Instantiate a reference object

You can instantiate a reference object, ie object-, document-, or user reference, using a smart constructor which takes one argument; either an external key or an id.

Java

```
// Smart constructors
ObjectReference objectReferenceFromExternalKey = new
    ObjectReference("EXTERNAL_KEY");
ObjectReference objectReferenceFromId = new ObjectReference(123456);
```

PL/SQL

```
-- Smart constructors
objref_from_external_key := ot_objectreference('EXTERNAL_KEY');
objref_from_id := ot_objectreference(123456);
```

Instantiate a document attribute

You can instantiate a document attribute using a smart constructor which takes an attribute reference and a value argument. The attribute reference may be specified as an external key or an object reference. The value may be a single value or a list of values for a multi value attribute. An implicit data conversion will take place if the value is of incorrect type.

Java

```
// Automatic data type conversion
DocumentAttribute clobAttribute = new DocumentAttribute("ATTRIBUTE_CLOB",
    "This string will be converted to a clob");
DocumentAttribute dimensionAttribute = new
    DocumentAttribute("ATTRIBUTE_CUSTOMER", "DIM_CUSTOMER_NSF");
dimensionAttribute = new DocumentAttribute("ATTRIBUTE_CUSTOMER", 12345);
```

```
DocumentAttribute multiValueDimensionAttribute = new
  DocumentAttribute("ATTRIBUTE_CUSTOMER",
    "DIMENSION_CUSTOMER_NSF", "DIMENSION_CUSTOMER_GARD");
```

PL/SQL

```
-- Automatic data type conversion
clob_docattr := ot_document_attribute('ATTRIBUTE_CLOB', 'This string will be
  converted to a clob');
dimension_docattr := ot_document_attribute('ATTRIBUTE_CUSTOMER',
  'DIM_CUSTOMER_NSF');
dimension_docattr := ot_document_attribute('ATTRIBUTE_CUSTOMER', 12345);
multival_dimension_docattr := ot_document_attribute('ATTRIBUTE_CUSTOMER',
  ct_document_attribute_values(ot_document_attribute_value('DIMENSION_CUSTOMER_NSF'),
  ot_document_attribute_value('DIMENSION_CUSTOMER_GARD')));
```

Document properties

You can handle document properties as plain attributes. Document properties are part of the document object, but may be created, updated, retrieved, and deleted using document attribute objects.

Java

```
// Document properties as plain attributes
DocumentAttribute titleAttribute = new DocumentAttribute("IKB$TITLE", "This
  is the document title");
DocumentAttribute documentTypeAttribute = new DocumentAttribute("IKB
  $DOCTYPE", "DOCTYPE_CUSTOMERINFO");

// Get document property - status
DocumentAttribute documentAttribute = ds.getDocumentAttribute(new
  UserReference("ORCLADMIN"), documentReference, new ObjectReference("IKB
  $STATUS"), GetOperationEnumeration.FULL);

// Update document property - title
ds.saveDocumentAttributes(new UserReference("ORCLADMIN"), documentReference,
  Arrays.asList(new DocumentAttribute("IKB$TITLE", "This is the updated
  title")),
  SaveOperationEnumeration.MERGE, DocumentVersioningModeEnumeration.NONE,
  null, null, false);
```

PL/SQL

```
-- Document properties as plain attributes
title_docattr := ot_document_attribute('IKB$TITLE', 'This is the document
  title');
document_type_docattr := ot_document_attribute('IKB$DOCUMENT_TYPE',
  'DOCTYPE_CUSTOMERINFO');

-- Get document property - status
l_status_docattr := ikb_service_api.get_document_attribute(
  ot_userreference('ORCLADMIN'),
  l_objref_document,
  ot_objectreference('IKB$STATUS'),
  ikb_service_api.get_operation_full
);

-- Update document property - title
l_objref_document := ikb_service_api.save_document_attributes(
  p_execution_user => ot_userreference('ORCLADMIN'),
  p_document_reference => l_objref_document,
  p_attributes => ct_document_attributes(
```

```

    ot_document_attribute('IKB$TITLE', 'This document is the updated title')
  )
);

```

Create document

You can create a document using the document object or simply by using document attribute objects.

Java

```

// Create document from a document object
Document document = new Document()
    .withTitle("This document is created from a java document object")
    .withDocumentTypeReference(new ObjectReference("TEMPDATA"))
    .withValidity(new Document.Validity(new GregorianCalendar(), null))
    .withContent("This is the text content")
    .withDocumentAttributes(
        new DocumentAttribute("SYSTEST_ATTR_CLOB", "This string will
be converted to a clob"),
        new DocumentAttribute("IKB_CUSTOMER",
Arrays.asList("DIM_CUSTOMER1", "DIM_CUSTOMER2"))
    );
documentReference = ds.saveDocument(new UserReference("ORCLADMIN"), document,
SaveOperationEnumeration.MERGE,
    SaveOperationEnumeration.MERGE, SaveOperationEnumeration.MERGE,
DocumentVersioningModeEnumeration.NONE,
    null, null, false);

// Create document from document attribute objects
documentReference = ds.saveDocumentAttributes(new UserReference("ORCLADMIN"),
null, // Document reference
    Arrays.asList(
        new DocumentAttribute("IKB$TITLE", "This document is created
from java document attribute objects"),
        new DocumentAttribute("IKB$DOCUMENT_TYPE", "TEMPDATA"),
        new DocumentAttribute("IKB$VALID_FROM", new
GregorianCalendar()),
        new DocumentAttribute("IKB$TEXT", "The text content"),
        new DocumentAttribute("SYSTEST_ATTR_CLOB", "This string will
be converted to a clob"),
        new DocumentAttribute("IKB_CUSTOMER",
Arrays.asList("DIM_CUSTOMER1", "DIM_CUSTOMER2"))
    ),
    SaveOperationEnumeration.MERGE, DocumentVersioningModeEnumeration.NONE, null,
null, false);

```

PL/SQL

```

-- Create document from a document object
l_document := ot_document();
l_document.title := 'This document is created from a ot_document object';
l_document.document_type_reference := ot_objectreference('TEMPDATA');
l_document.valid_from := sysdate;
l_document.text_content := 'This is the text content';
l_document.attribute_references := ct_document_attributes(
    ot_document_attribute('SYSTEST_ATTR_CLOB', 'This string will be converted
to a clob'),
    ot_document_attribute('IKB_CUSTOMER', ct_document_attribute_values(
        ot_document_attribute_value('DIM_CUSTOMER1'),
        ot_document_attribute_value('DIM_CUSTOMER2'))))

```

```

);
l_objref := ikb_service_api.save_document(
  p_execution_user => ot_userreference('ORCLADMIN'),
  p_document       => l_document
);

-- Create document from document attribute objects
l_objref := ikb_service_api.save_document_attributes(
  p_execution_user      => ot_userreference('ORCLADMIN'),
  p_document_reference => ot_document_reference(),
  p_attributes          => ct_document_attributes(
    ot_document_attribute('IKB$TITLE', 'This document is created from
ot_document_attribute objects'),
    ot_document_attribute('IKB$DOCUMENT_TYPE', 'TEMPDATA'),
    ot_document_attribute('IKB$VALID_FROM', sysdate),
    ot_document_attribute('IKB$TEXT', 'The text content'),
    ot_document_attribute('SYSTEMEST_ATTR_CLOB', 'This string will be converted
to a clob'),
    ot_document_attribute('IKB_CUSTOMER', ct_document_attribute_values(
      ot_document_attribute_value('DIM_CUSTOMER1'),
      ot_document_attribute_value('DIM_CUSTOMER2')
    ))
  )
);

```

Delete document attribute

You can delete a document attribute from a document. You can also choose to delete only a single value of a multi value document attribute.

Java

```

// Delete document attribute value
ds.saveDocumentAttributes(new UserReference("ORCLADMIN"), documentReference,
  Arrays.asList(new DocumentAttribute("IKB_CUSTOMER",
  Arrays.asList("DIM_CUSTOMER1"))),
  SaveOperationEnumeration.DELETE_VALUE,
  DocumentVersioningModeEnumeration.NONE,
  null,
  null,
  false);

```

PL/SQL

```

-- Delete document attribute value
l_objref_document := ikb_service_api.save_document_attributes(
  p_execution_user      => ot_userreference('ORCLADMIN'),
  p_document_reference => l_objref_document,
  p_attributes          => ct_document_attributes(
    ot_document_attribute('IKB_CUSTOMER', 'DIM_CUSTOMER1')
  ),
  p_attribute_operation => ikb_service_api.save_operation_delete_value
);

```

17. iKnowBase Process Services for Activiti

The iKnowBase Process Services for Activiti is a set of tools and techniques that enables the building of process-centric applications using the combined power of iKnowBase® and Activiti®. The iKnowBase Process Services for Activiti comprises the following items:

- An embedded Activiti process engine, inside the iKnowBase `ikbViewer` application
- Extensions to the Activiti engine, providing easy access to the iKnowBase repository from Activiti `ScriptTasks`
- Extensions to the iKnowBase engine, providing easy access to the Activiti process engine from iKnowBase scripts
- An “Activiti form” portlet, providing Activiti form display capabilities from iKnowBase applications
- iKnowBase Process Studio (optional iKnowBase web application plugin), a web-based tool for starting and monitoring processes and completing user tasks (mainly intended as a demo and developer application)
 - Also contains Activiti process samples that can be executed in iKnowBase Process Studio

Preparations

In order to use the iKnowBase Process Services for Activiti, the following must be done:

- The Activiti engine must be enabled. This is described in the *Installation Guide*, in the chapter `Viewer` module.
- Developers must download Eclipse, and install the Activiti designer plugin. For more information, see the <http://www.activiti.org> web site.
- Install iKnowBase Process Studio plugin. This module was automatically installed in iKnowBase < 7.0 but is now an optional plugin. See installation guide for details regarding plugin installation.
- Optionally, the Activiti process samples may be deployed. These are available for download from the resource directory and easily available using the Help-link in iKnowBase Process Studio.

Developing process applications

Developing process applications typically comprises the following steps:

- Develop the BPMN process, using Activiti Designer:
 - Wherever you want to have user interaction, set up a `UserTask` in Activiti. Declare the form items you want to edit in the process model.
 - Save the BPMN file to disk, give it the extension `.bpmn`, and install into the repository: `iknowbase <propertyfile> deployActivitiResource <bpmn file>` or by using iKnowBase Process Studio
- Develop the iKnowBase applications, using the iKnowBase Development Studio:
 - Use the `Template Viewer` portlet to load information from Activiti, for things such as inboxes and process overviews (presentations, generally).
 - Use the `Activiti Form` portlet to configure forms that start Activiti processes and complete Activiti `UserTasks`.
 - Use the `Script Target` to implement custom controllers.

The iKnowBase applications may also be implemented in Java.

The integration mechanisms between the two technologies (iKnowBase and Activiti) are kept intentionally simple:

- From Activiti, the `Activiti ScriptTask` has access to a script object called “`iknowbase`”, which provides access to the iKnowBase `ServiceAPI` and database
- From iKnowBase, any Groovy template (the `Template Viewer` and the `Script Target` in particular) has access to a script object called “`bpmClient`”, which provides access to the Activiti API

Your process data will be available in iKnowBase Process Studio.

Accessing Activiti data from a Template Viewer

This groovy script shows the fundamentals of accessing and presenting data from an iKnowBase component:

```
// Prepare some utility variables
def engine = bpmClient.activitiEngine

// Create a query using Activiti apis, and load data
def query =
    engine.historyService.createHistoricProcessInstanceQuery().startedBy("orcladmin").orderByProcessDefinitionId()
def rowcount = query.count()
def instances = query.listPage(1, 20)

// Render data using the groovy MarkupBuilder which is automatically supplied
html.div {
    h1 ("Process instances started by orcladmin, ordered by processInstanceId
(rows 1-20)")
    table {
        tr {
            th ("Process Definition Id")
            th ("Process Instance Id")
            th ("Start time")
        }
        instances.each { instance ->
            tr {
                th (instance.processDefinitionId)
                th (instance.id)
                th (instance.startTime)
            }
        }
    }
}
```

Accessing iKnowBase data from a ScriptTask

This groovy script shows the fundamentals of accessing and working with iKnowBase data from an Activiti ScriptTask:

```
import com.iknowbase.api.contentservices.v2.model.*

// Init
def userref = new UserReference().withUsername(initiator)
def docref = new DocumentReference().withId(ikbDocumentId)

def document = iknowbase.documentService.getDocument(
    userref,
    docref,
    GetOperationEnumeration.MIN,
    GetOperationEnumeration.NONE,
    GetOperationEnumeration.NONE
)

document.setDescription(document.getDescription() + "<p>Activiti process:
Document approved</p>")

iknowbase.documentService.saveDocument(
    userref,
    document,
    SaveOperationEnumeration.MERGE,
    SaveOperationEnumeration.NONE,
```

```

    SaveOperationEnumeration.NONE,
    null, null, null, false
)

```

Using a form to start processes and complete UserTasks

First, define an Activiti StartEvent or UserTask with the relevant form properties. The XML below is an extract from the Activiti BPMN definition:

```

<userTask id="registerissue" name="Register issue"
  activiti:assignee="{initiator}"

  activiti:formKey="urn:iknowbase:form:guid:C26AD8CA297355DCE040000A180062E2">
  <extensionElements>
    <activiti:formProperty id="hdTitle" name="Title" type="string"
      required="true"></activiti:formProperty>
    <activiti:formProperty id="hdDescription" name="Description"
      type="string"></activiti:formProperty>
    <activiti:formProperty id="hdPriority" name="Priority" type="long"></
activiti:formProperty>
    <activiti:formProperty id="hdCategory" name="Category" type="long"></
activiti:formProperty>
  </extensionElements>
</userTask>

```

Then, in iKnowBase, define a page with an Activity Task Form component based on a FreeMarker template such as this one:

```

[#macro showForm]
<h2>Comments</h2>
[@form.form]
  <table>
  <tr>
    <td>[@form.label bind="task.hdTitle" /]</td>
    <td>[@form.input name="task.hdTitle" /]</td>
  </tr>
  <tr>
    <td>[@form.label bind="task.hdDescription" /]</td>
    <td>[@form.input name="task.hdDescription" type="textarea" /]</td>
  </tr>
  <tr>
    <td>[@form.label bind="task.hdPriority" /]</td>
    <td>[@form.input name="task.hdPriority" type="select"
attribute="IKB_PRIORITY" /]</td>
  </tr>
  <tr>
    <td>[@form.label bind="task.hdCategory" /]</td>
    <td>[@form.input name="task.hdCategory" type="select"
attribute="IKB_SUBJECT" /]</td>
  </tr>
  </table>
  [@form.button name="submit"          action="submit"
attributes='accesskey="x"'          ]Complete task[/@form.button]
[/@form.form]
[/#macro]

[#macro thankyou]
<h1>Thank you!</h1>
<p>Your comment has been submitted. Click to view
<a href="/dev/activiti/processdefinitions/processinstance?processInstanceId=
${form.data["task"].processInstanceId}">process details</a>.</p>

```

```
[/#macro]

[#if form.currentCommand.action! == 'submit']
    [@thankyou /]
[#else]
    [@showForm /]
[/#if]
```

A couple of items of interest:

- The Activiti UserTask defines a “formKey” attribute, which could be used to reference the target (e.g. referring to the guid of the FreeMarker template)
- The iKnowBase Activiti Form uses the FreeMarker directives `form.form`, `form.label`, `form.input` and `form.button` to generate the form
- The template form items use a name on the format “task.<propertyname>” to refer to the UserTask properties
- The template items “task.hdPriority” and “task.hdCategory” in the example refer to an iKnowBase attribute to get a proper control with proper defaults
- The template is used both for initial rendering and after submit, and uses the test “form.currentCommand.action! == 'submit'” to detect the phase

See the chapter Activiti Form Viewer in *iKnowBase API Reference* for further details.

Associating an Activiti Process with an iKnowBase Document

For associating an Activiti process instance with an iKnowBase document, we recommend that you use a process instance variable named “documentId” of type “long”. iKnowBase Process Studio provides filtering mechanisms for this variable.

18. iKnowBase Development Toolkit

The iKnowBase Development Toolkit enables you to develop iKnowBase Plugins in your IDE and assemble a customer specific edition of the iKnowBase application including the required plugin changes.

While the development toolkit does not replace the standard iKnowBase declarative development model, it provides the developer with a supplement and alternative.

Getting started

Developers are strongly encouraged to participate in a workshop with the iKnowBase product development team before starting with the Development Toolkit.

Read the other chapters after “Getting started” for detailed information about iKnowBase plugins and technology.

About the build system

The Development Toolkit uses the *Gradle build system* for

- build logic
- project structure
- managing project dependencies
- start from command line (or some tool invoking the Gradle build)
- assemble from command line (or some tool invoking the Gradle build)

Gradle is supported by several of the major IDEs, enabling you to import the Gradle project into the IDE.

Requirements

You should already have (or do it now):

- Java SE Development Kit matching your target environment (see release notes for supported Java version)
- The IDE of your choice (the sample build system requires gradle support)
- Downloaded iKnowBase distribution files for development (see next section)
- Database access to an installed iKnowBase Database Repository

Distribution files for development

The iKnowBase distribution files for development are

- sample-build-root-7.1.7.zip
- sample-plugin-projects-7.1.7.zip

sample-build-root

The sample-build-root is a complete example of a Gradle setup for plugin development. You may use it as is, or develop your own build based on elements in the sample.

- gradle (directory): gradle wrapper for bootstrapping the build system
- gradle.properties: build properties
- iknowbase.properties: iKnowBase properties
- settings.gradle: project hierachy declarations

sample-plugin-projects

The sample-plugin-projects zip contains a set of technology samples to help you get started developing plugins for iKnowBase.

Step-by-step setup

1. Extract the provided development .zip files

Start with the sample-build-root to create a build workspace. If you want to add the sample projects, we suggest an extraction directory structure like this:

```
sample-build-root
  sample-plugins
    // contents of sample-plugin-projects here
```

2. Configure build

gradle.properties

We use *Artifactory* for dependency resolution for both external 3rd party and iKnowBase artifacts.

The gradle.properties file is by default configured with the required settings for accessing artifactory.

settings.gradle

Add your plugin projects here. The file contains examples for including the iKnowBase samples as well as set up your custom plugins.

When adding your own plugins, you need to create the project structure that goes with it. The directory structure depends on the type of plugins you create. Each project should contain a build.gradle file (see next section for build.gradle description).

Example:

For the project definition in settings.gradle:

```
include 'custom-plugins'
include 'custom-plugins:custom-java-jar-plugin'
include 'custom-plugins:custom-groovy-jar-plugin'
include 'custom-plugins:custom-java-war-plugin'
```

You need to create the project structure:

```
sample-build-root
  custom-plugins
    custom-java-jar-plugin
      build.gradle
      src
        main
          java
          resources
    custom-groovy-jar-plugin
      build.gradle
      src
        main
          groovy
          resources
    custom-java-war-plugin
      build.gradle
      src
        main
          java
          resources
          webapp
```

build.gradle for your plugin projects

We recommend using a build.gradle file per project as shown above. The build.gradle will set up plugin project specifics such as:

- Build plugins (java, groovy, war)
- Dependencies

For a java (.jar) project:

```
apply plugin: 'java'

dependencies {
    provided "com.iknowbase:iknowbase-webapp:$ikbVersion"
    provided ikbLibs.servlet
}
```

For a groovy (.jar) project:

```
apply plugin: 'groovy'

dependencies {
    provided "com.iknowbase:iknowbase-webapp:$ikbVersion"
    provided ikbLibs.servlet
}
```

For a java web application (.war) project:

```
apply plugin: 'java'
apply plugin: 'war'

dependencies {
    // use 'providedCompile' instead of 'provided' for war projects
    providedCompile "com.iknowbase:iknowbase-webapp:$ikbVersion"
    providedCompile ikbLibs.servlet
}

// Customized war that will add module source as jar to WEB-INF/lib instead
// of the usual WEB-INF/classes
// This avoids name collision for war resources with the same name (like
// "spring.factories") when patching
// with multiple war files
war {
    // First we'll remove the main build output from the classpath
    classpath = classpath.filter {!(it.name.equals('main'))}
    // And then add the module's jar archive
    classpath jar
}
```

See sample projects for relevant build.gradle content.

build.gradle for your root project

Include your plugin projects in the dependencies section (NOT in buildscript dependencies!).

3. Configure iKnowBase properties

- iknowbase.properties:

- Set db.URL, db.ikbUser and db.ikbPassword to match your existing iKnowBase database repository

4. Test from command line

Start a command prompt in the sample-build-root.

The first gradle command entered will trigger download of the gradle binaries and iKnowBase dependencies.

To provide a list of build tasks available to you:

```
gradlew tasks
```

To list all projects in your build:

```
gradlew projects
```

To start the iKnowBase web server with all plugins (that you've added to the root project runtime) on localhost port 80:

```
gradlew ikbRun
```

You can access the iKnowBase installation at <http://localhost> and verify that everything is up and running.

5. Import in your favorite IDE

Now that you know that the base build is OK, it's time to import to an IDE and start development.

Note: If you change gradle build scripts (settings.gradle, build.gradle) you may need to re-import / refresh to get the updates in your IDE.

IntelliJ IDEA

Open the sample-build-root/build.gradle. The IDE will import and setup the project for you.

Eclipse

Eclipse requires additional gradle plugin: <https://github.com/spring-projects/eclipse-integration-gradle/>. Once installed, import the sample-build-root as a gradle build. After import select all projects and adjust project properties for Gradle: Project > Properties > Gradle: Classpath sorting strategy: As returned by build script

6. Setup iKnowBase web server run configuration in IDE

Note: If you skipped "ikbRun" step on the command line, you must run the gradle task "ikbPrepare" before starting.

To start iKnowBase with all plugins from the IDE, add an application run configuration with:

- Classpath of root project (sample-build-root)
- Main class: com.iknowbase.jetty.util.QuickstartFactory
- Program args: iknowbase.properties

Assemble your plugins

All projects may be assembled using:

```
gradlew assemble
```

The resulting jar/war will be generated in <project dir>/build/libs/.

Patch iKnowBase web application with your plugins

When you want to deploy iKnowBase with custom plugins to test/production, you need to create a new patched .war file. The iKnowBase wrapper script (iknowbase.sh|.bat) is recommended for adding assembled plugins to your existing iknowbase-webapp-7.1.7.war.

Given the standard iKnowBase installation directory structure (See *Quick installation and upgrade overview*):

```
iknowbase-7.1.7
  // extracted distribution files

production (named runtime directory)
  iknowbase.properties
  iknowbase.sh

plugins
  // all plugins should go here
  plugin-1.jar
  plugin-2.war

wars
  // your new patched .war files should go here
```

Assemble your new war with `./iknowbase.sh <config> assembleWar <input> <output> <extensions>`:

```
cd production
./iknowbase.sh iknowbase.properties assembleWar ../iknowbase-7.1.7/
wars/iknowbase-webapp-7.1.7.war ./wars/iknowbase-webapp-7.1.7-custom.war ./
plugins/*
```

Run using the iKnowBase web server

Custom .war files, like the one you just created, must explicitly be specified in the iKnowBase properties file.

An alternative for .jar plugins when target server is the iKnowBase web server , is using the `web.extraClasspath` setting instead of patching the .war file. Point it to your jar files and they will be available in the classpath when the server has started.

```
web.extraClasspath=./plugins/*.jar
```

Deploy

Deploy the new `iknowbase-webapp` war file to you application server. No special requirements.

Plugin assembly

By using the toolkit, you'll be able to develop your own as well as include externally developed plugins.

Once developed, a plugin will be assembled to one of two forms:

- .jar (Java archive)
- .war (Web archive) (may contain .jar files)

.jar plugin

A .jar plugin contains

- compiled classes
- resources

and is the typical choice if you do not require additional libraries.

You may still use the .jar plugin if you require additional libraries, but you must then collect all the library .jars yourself when patching the iKnowBase application war.

build.gradle:

```
    apply plugin: 'java' // and/or 'groovy', if you prefer groovy development

dependencies {
    provided "com.iknowbase:iknowbase-webapp:$ikbVersion"
}
```

.war plugin

A .war plugin contains:

- compiled classes
- resources
- library dependencies
- web application configuration
- web application resources

and is the typical choice if you need more than what is offered through a .jar.

Note: The content in the resulting plugin .war file will override the existing content in the source iKnowBase .war file when patching.

build.gradle:

```
    apply plugin: 'java' // and/or 'groovy', if you prefer groovy development
    apply plugin: 'war'

dependencies {
    // use 'providedCompile' instead of 'provided' for war projects
    providedCompile "com.iknowbase:iknowbase-webapp:$ikbVersion"
}

// Customized war that will add module source as jar to WEB-INF/lib instead
// of the usual WEB-INF/classes
// This avoids name collision for war resources with the same name (like
// "spring.factories") when patching
// with multiple war files
war {
    // First we'll remove the main build output from the classpath
    classpath = classpath.filter {!(it.name.equals('main'))}
    // And then add the module's jar archive
    classpath jar
}
```

Libraries

Note: All libraries in use by the iKnowBase web application are, due to the current build system, available to your plugin at both compile and runtime. However, all “com.iknowbase” packages except “com.iknowbase.api” are considered internal to iKnowBase and use of these packages are prohibited and without support.

All dependencies are managed in your project’s build.gradle file.

The plugin projects will get all libraries distributed through the iKnowBase application with this dependency:

```
    // use one of
    provided          "com.iknowbase:iknowbase-webapp:$ikbVersion" //for
groovy/java
    providedCompile  "com.iknowbase:iknowbase-webapp:$ikbVersion" //for war
(even if you java groovy/java plugin activated)
```

Remember to also add provided dependency on servlet API as this dependency will not be provided through iknowbase-webapp.

Servlet API is normally provided by the application/web server:

```
bc.  
// use one of  
provided ikbLibs.servlet  
providedCompile ikbLibs.servlet
```

The project may also bring additional libraries, **provided they are not conflicting with the existing libraries**.

Additional libraries will automatically be included in the resulting plugin .war file when using the .war plugin.

The iKnowBase application container

The iKnowBase application is a java web application based on the Spring Framework (<http://projects.spring.io/spring-framework/>). Your code will need to be deployed into this application, and can then be used from several points and for several purposes.

Extension points

The table shows what type of extensions and modifications are available

Extension type	Language
Spring Configuration	Java, Groovy
Spring Bean	Java, Groovy
Spring MVC Controller	Java, Groovy
Spring MVC View	Java, Groovy, Freemarker
Spring MVC Static plugin-resources	N/A
Java extension	Java
Java Activiti extension	Java
Groovy extension	Groovy
Freemarker extension	Freemarker
Log4j configuration	Log4j
Web application override	Any

Configuring the application container

The Spring container keeps track of java beans for the lifetime of the iKnowBase application, and many plugins will need to provide their own beans. To expose your beans, you will need to create and expose a configuration class:

- Create your own Spring @Configuration class, responsible for creating @Beans (and other configuration tasks)
- Create a file "META-INF/spring.factories" containing information about the configuration class

In your source code, this typically looks like this:

- src/main/java/YOUR_PACKAGE/YOUR_SPRING_CONFIGURATION_MODULE (or /src/main/groovy/...)
- src/main/resources/META-INF/spring.factories

The Spring Configuration classes are plain Java/Groovy classes annotated with @Configuration (org.springframework.context.annotation.Configuration). A configuration class allows Spring Bean generation and web application context customizations. The class contains methods annotated with @Bean (org.springframework.context.annotation.Bean) where the resulting bean name is the name of the method and the bean type is the declared and returned object.

To load the Spring Configuration class, create `src/main/resources/META-INF/spring.factories` in your plugin project with:

```
com.iknowbase.api.server.SpringModule=<YOUR_PACKAGE/
YOUR_SPRING_CONFIGURATION_MODULE>
```

Spring @Beans

Spring beans are objects managed by the Spring container. Beans are generated using the Spring factory pattern described in the “Spring plugin bootstrap” section above.

Inject services

The table below shows the iKnowBase specific services that you can @Autowire in your Spring beans, typically in you @Configuration class. Note that these services are not available as resolved method arguments in a Spring Controller.

Service	Description
<code>com.iknowbase.api.contentservices.ContentServicesEngine</code>	<i>iKnowBase Content Services API</i> for accessing the iKnowBase ContentServices engine.
<code>com.iknowbase.presentation.services.PresentationServicesEngine</code>	<i>iKnowBase Presentation Services API</i> for accessing the iKnowBase PresentationServices engine.

The services below are available directly for backward compatibility reasons. They can (and should) be accessed through the ContentServicesEngine or PresentationServicesEngine above.

Service	Description
<code>javax.sql.DataSource</code>	DataSource for accessing the iKnowBase Database Repository.
<code>com.iknowbase.api.contentservices.v2.service.Docu</code>	<i>iKnowBase Content Services API</i> for accessing the iKnowBase Content Repository.

Inject configuration

The iKnowBase applications can be configured as described in *Installation Guide > Configuration*. You may also add your own configuration keys to this set.

The configurations can be injected in the @Configuration’s @Bean methods using @Value-annotation. You pass the values to your actual bean when constructing the bean.:

Configuration option	Default value
@Value("\${JAVA_HOME}") String javaHome	not set
@Value("\${JAVA_HOME:#{null}}") String javaHome	null
@Value("\${JAVA_HOME:null}") String javaHome	“null”
@Value("\${com.iknowbase.pageEngine.contentCacheEnabled}") boolean contentCacheEnabled	false
@Value("\${thisKeyShouldNotBePresentUseDefault:defaultvalue}") String thisKeyShouldNotBePresentUseDefault	defaultvalue

If a default value has not been specified, the configuration MUST exist.

Spring MVC Controller

The *Spring MVC Controller* is used to expose your extension at the specified web request path. See the Spring documentation and the provided iKnowBase Development Toolkit samples.

The concept of using a Spring MVC Controller is as follows:

- Make sure you have a @Configuration class, so that Spring can find your code.

- Create a java-class (a controller), annotated with @RequestMapping.
- Create an instance of this java-class in your @Configuration class.
- On startup, the application will first discover the @Configuration class and create the @Beans it specifies. It will then look at those @Beans, and discover @RequestMappings.
- At runtime, the application will look for a @Bean with a @RequestMapping that matches the request, and run it.

NOTE: DO NOT use the @Controller annotation. Construct your controller bean in the @Configuration class and annotate the Controller itself with @RequestMapping, if you require @RequestMapping support.

Inject services and configurations

Since a Spring MVC Controller is just a regular Spring Bean, you can inject services and configurations as for any @Bean.

Inject method arguments

When Spring runs your @RequestMapping method, it will automatically pass parameter values to that method. In addition to the default argument resolving available for Spring MVC (like HttpServletRequest, HttpServletResponse, etc), the iKnowBase web application supports its own set of services and resolved arguments.

The following table shows the iKnowBase specific services available to you in the Spring configuration and controller classes and where it can be injected.

Service	Description
com.iknowbase.api.contentservices.ContentServicesClient	<i>iKnowBase Content Services Client</i> for accessing the iKnowBase Content Services.
com.iknowbase.api.presentation.services.PresentationServicesClient	<i>iKnowBase Presentation Services Client</i> for accessing the iKnowBase Presentation Services.
com.iknowbase.api.presentation.services.client.ActivitiClient	<i>iKnowBase Activiti Client</i> for accessing the Activiti process engine.
com.iknowbase.api.presentation.services.client.SolrSearchClient	<i>iKnowBase Solr Search Client Provider</i> for accessing the configured Solr Search Clients.
com.iknowbase.api.presentation.services.client.SpringMvcPageClient	<i>iKnowBase Spring MVC Page Client</i> for accessing the iKnowBase Page Engine.

Security

Controllers may be annotated with Spring Security annotations if you want to restrict access to a controller method.

Some examples of standard expressions included with Spring Security:

Configuration option	Default value
@PreAuthorize("isAnonymous()")	Access as public user
@PreAuthorize("isAuthenticated()")	Access as non-public user

In addition to the standard Spring Security annotations, the following iKnowBase specific annotations are supported:

Configuration option	Default value
@PreAuthorize("hasPermission('[ACL_EXTERNAL_KEY]', membership and privilege READ 'ACL', 'READ')")	
@PreAuthorize("hasPermission('[ACL_EXTERNAL_KEY]', membership and privilege MODIFY 'ACL', 'MODIFY')")	
@PreAuthorize("hasPermission('[ACL_EXTERNAL_KEY]', membership and privilege DELETE 'ACL', 'DELETE')")	

@PreAuthorize("hasPermission('executionUser', 'ADMIN')")	Current effective user has administrator privileges
@PreAuthorize("hasPermission('authenticatedUser', 'ADMIN')")	Real authenticated user has administrator privileges

executionUser and authenticatedUser will normally be the same, but if an administrator has switched to a different user in the current session, the authenticatedUser will be the administrator and the executionUser will be the switched to user.

Static web resources

Static web resources, such as css, javascript and images, can be included in your plugin project. The resources will be assembled and served from jar files.

- Directory in plugin source tree: `src/main/resources/META-INF/resources/plugin-resources`
- Directory in assembled jar file: `META-INF/resources/plugin-resources`
- Web path: `//server/application-root/plugin-resources`

Note! The static resources namespace will be shared among all plugins. Be sure to use plugin specific directories for your static content.

Alternatively, static web resources may be served as an additional war project of your choosing, like `/custom-resources`. The `/ressurs` is by default taken by the `iknowbase-resources`. See `iknowbase.properties` if you need to override this behavior.

Activiti

Activiti processes can be extended using *Execution Listeners*, *Task Listeners* and *Service Tasks*.

The *iKnowBase Repository Model* is accessible from task and execution listeners and service tasks, provided is has been mapped in the process definition.

Using a plugin from iKnowBase Development Studio groovy components

Java or Groovy classes may be used from groovy components you define in iKnowBase Development Studio. This enables you to create reusable components in plugins and access them in both the plugin and the traditional development model.

Logging configuration

Log4j configuration files are present under `WEB-INF/classes` in the iKnowBase application and you may add additional configurations or override configurations. When overriding, make sure the plugin war has the configurations files defined in `WEB-INF/classes` as files added here take presedence over files found in library files in `WEB-INF/lib`.

You may also start the iKnowBase web server with a specific log4j configuration by adding the jvm option `-Dlog4j.configuration=file://<your configuration file>` to the run configuration in your IDE.

Solr Server

The iKnowBase application server provides a `SolrClient`-object for easy access to the search functionality of a Solr server. This is often configured using the installation properties of an installation, but if you need more control of the `SolrServer` connection, you can use the Development Toolkit to configure the server yourself:

```
@Bean(name="acmeInternetSearch")
public SolrServer createSolrServer() {
    CloudSolrServer cloudSolrServer = new
    CloudSolrServer("zookeeper.solr.acme.com");
    cloudSolrServer.setParser(new XMLResponseParser());
    return cloudSolrServer;
}
```

When the above @Bean is exposed to the application container (through a @Configuration class), iKnowBase will automatically discover it, and use it as the basis of a search connection. The name specified in the @Bean will be used for the search server.

Form

iKnowBase simplifies form development that targets iKnowBase documents with

- Bean-to-document mapping
- Form Processor Macros
- Classic integration (External Forms)

See sample: sample-plugin-form

Bean-to-document mapping

To easily convert between an iKnowBase document and a standard Java Bean (Not a Spring Bean), iKnowBase provides annotations for use in the Java Bean that enables automatic conversion and efficient development.

Annotation	Description
@AttributeMapping	For mapping iKnowBase attributes to properties. Processed by the BeanProcessor class. Use on getter methods.
@Init	For methods that needs to be invoked during initialization of the form backing bean.
@BindingError	For methods that needs to be invoked after binding and validation of the form backing bean.
@BeforeBeanToAttributesMapping	For methods that needs to be invoked before a bean is converter to DocumentAttributes.
@BeforeInsert	For methods that needs to be invoked after conversion to document attributes and before save to iKnowBase repository as new document.
@BeforeUpdate	For methods that needs to be invoked after conversion to document attributes and before save to iKnowBase repository updating an existing document.
@AfterLoad	For methods that needs to be invoked after load from iKnowBase repository.
@AfterInsert	For methods that needs to be invoked after save of new document to iKnowBase repository.
@AfterUpdate	For methods that needs to be invoked after save of existing document to iKnowBase repository.

Form Processor

The iKnowBase Presentation Services Form Processor aids with conversion, loading, insert and update of iKnowBase AttributeMapping annotated beans (see previous section). It also provides *FreeMarker macros* to ease integration with the Java Bean and iKnowBase Metadata.

Form integration with iKnowBase Classic components

An iKnowBase Classic Form may now also be configured as an External Form. An External Form supports mapping against a URL path, where the plugin containing the requested form is available.

If you store the External Form reference on the document, the plugin form will be used in generic iKnowBase Classic edit links.

An External Form may also be connected to document types and set as default form to allow using the plugin form on documents that have no specific form registration.

Normal Quicklink and Target integration for forms also applies to an External Form.

You may convert a Clasic Form component between Classic Form and External Form and thus keep the preexisting form GUID used on the documents.

Java based versions of News Article and Admin Form are available in iKnowBase Content Studio and External Form data has been registered. The form is classic by default, but may easily be converted to an External Form supported by Java. See source code for the forms and their required metadata.

Available samples

The following samples are included when installing the Development Toolkit:

Sample name	Description
sample-plugins:sample-activiti-controllers	Demonstrates using DataSource, DocumentService and ActivitiClient.
sample-plugins:sample-activiti-process-definitions	Demonstrates a project containing Activiti process definitions. Also demonstrates how to map the iKnowBase RepositoryModel, listeners and service task.
sample-plugins:sample-activiti-extensions	Demonstrates task and execution listeners for Activiti. Also demonstrates how to use the iKnowBase RepositoryModel.
sample-plugins:sample-plugin-config	Demonstrates accessing iKnowBase Configuration options.
sample-plugins:sample-plugin-freemarker	Demonstrates various ways of using freemarker as presentation view technology.
sample-plugins:sample-plugin-groovy	Demonstrates writing controllers in groovy and use freemarker as presentation view technology as well as various ways of using groovy as presentation view technology.
sample-plugins:sample-plugin-hello	Demonstrates a very simple controller for a minimal start.
sample-plugins:sample-plugin-msoffice	Demonstrates using a third party library for generating Excel files.
sample-plugins:sample-plugin-static-content	Demonstrates using static content, such as css, js and images.
sample-plugins:sample-plugin-form	Demonstrates developing forms. See form section for details.

iKnowBase Reference Plugins

While the samples listed above are focused on demonstration of technology, the iKnowBase distribution includes two complete real life plugins that you may use as reference. The source code is provided alongside the distribution.

Sample name	Description
iknowbase-content-studio	The iKnowBase Content Studio (/cs).
iknowbase-process-studio	The iKnowBase Process Studio (/ps).

Troubleshooting

Classpath issues in Eclipse based IDEs

When importing the iknowbase-plugins project with subprojects in Eclipse, the classpath defaults generated by the eclipse plugin is not correctly ordered.

Resolve the issue with

- During Gradle project import: Uncheck "Enable dependency management"
- For all IDE helper projects:
 - Project Properties > Gradle > Classpath sorting strategy: Select "As returned by build script"

Out of memory issues when building gradle project model

You may encounter issues when building the gradle project model during import in Eclipse. Adjust your `gradle.properties` file and enable

```
org.gradle.jvmargs=-Xms128m -Xmx256m
```

Accented (Norwegian) text loaded using resource bundle has wrong encoding

Make sure the property files (*.properties) are located in source tree 'src/main/resources' and NOT 'src/main/java'. Encode all accented characters using unicode notation "\u<code>".

My Spring MVC Controller is not available as expected on the mapped path

iKnowBase will NOT do any Spring component scanning. To load a Spring controller and expose the methods you need to

- @Configuration class with @Bean method constructing the controller bean.
- spring.factories needs to point to your @Configuration class.
- Controller needs to be annotated with @RequestMapping on the class level
- The controller's methods you want to expose needs to have it's own @RequestMapping annotation

NOTE: iKnowBase versions up to and including 6.6 had component scan for the "com.iknowbase.plugin" package. This has been removed and replaced with the spring.factories and @Configuration.

Run from IDE does not start iKnowBase. Seems like application is missing.

The "ikbPrepare" gradle task downloads and unpacks all required iKnowBase web app resources. If you skipped "ikbRun" step on the command line, you must run the gradle task "ikbPrepare" before starting. "ikbRun" will call "ikbPrepare".

19. User Administration

Overview of User Administration

User administration is the task of managing which users and groups are known to the iKnowBase system. Generally, being known to the system means that you have access to at least parts of the system.

iKnowBase has two key concepts for user administration:

- Users define the principals known to iKnowBase. Generally, a principal is the same as a person, but it may also be another computer system which has access to iKnowBase.
- Groups are collections of users. By itself, this has no meaning, but groups are useful later on, for assigning permissions to more than one user.

Manage Users

Automatic User Synchronization

User and group definitions are often defined in other systems, such as Microsoft Active Directory or another user directory.

In such cases, use an integration mechanism for automatically transferring user information:

- The iKnowBase LDAP synchronization mechanism will transfer users and groups from an LDAP (Oracle Internet Directory, OpenDJ or Microsoft Active Directory). Use the iKnowBase Development Studio to configure the synchronization jobs.
- The iKnowBase Service API has endpoints for adding, deleting or changing users and groups, as well as groups and group memberships. Use this if you want automatic synchronization from other users sources.

Manual User Synchronization

When the set of users is not defined elsewhere, or when you want to manage extra users, use the **user directory** tools of the iKnowBase Development Studio to perform the required changes. See *User Reference* for further information.

Manage Groups

A core set of groups are often synchronized from other systems, but it is also common to add iKnowBase-specific groups directly to the iKnowBase repository. Here, the business requirements will often decide what you need to do.

The following set of groups is often useful:

- An administrator group, with permissions to most metadata and content.
- Groups for information owners, with access to their own defined areas.
- A group for everybody, with access to all internal content (for intranet usage).
- A group for external user, with access to all web content (for www.example.com).
- A group for managers.
- A group for each project.

Use the **user directory** tools of the iKnowBase Development Studio to manage groups. See *User Administration Reference* for further information.

20. Security Administration

Overview of Security Administration

Security administration is the task of deciding what users have access to what objects. Here, you will:

- Manage Access Control Lists, including members and their permissions.
- Assign Access Control Lists to certain objects.

iKnowBase uses Access control lists (ACLs) to manage permissions:

- An ACL is a set of groups and users, referred to as members, that are assigned permissions. The ACL itself does not, however, decide which documents or other objects the permissions apply to.
- Later, the ACL is used by objects (such as a document), and the permissions specified by the ACL is applied to the object.

Access control list member privileges:

Property	Description
Read	Granted the privilege to read content in iKnowBase.
Modify	Granted the privilege to read and modify content in iKnowBase.
Protected	Granted the privilege to view dimensions that are protected with the access control list in iKnowBase.
Delete	Granted the privilege to delete content in iKnowBase.
Approver	Granted the privilege to approve the publication of content in iKnowBase.
Creatable	Granted the privilege to create content in iKnowBase.
Categorize	Granted the privilege to categorize content with a dimension that is protected with the access control list.
Valid from	The date from which the member is valid.
Valid to	The expiry date for the member.

Manage Access Control Lists

The ACLs are maintained in iKnowBase. Use the **user directory** tools of the iKnowBase Development Studio to manage ACLs. See *User Administration Reference* for further information.

Assign Access Control Lists to Objects

ACLs may be assigned to the following objects:

- Information objects, also referred to as documents
- Dimensions
- Forms
- Task Wizards
- Pages
- Subsystems

You may assign an ACL to a document when it is published or by modifying an existing document using a form.

You may assign an ACL to a dimension by using the **metadata management** tools of iKnowBase Development Studio, see *User Administration Reference* for further information.

You may assign an ACL to a form, task wizard, page, or subsystem by using the **development tool** tools of iKnowBase Development Studio, see *Development Reference* for further information.

Overview of Document Access

Every iKnowBase document (information object) may have an ACL applied to it. This ACL decides which users are given permission to the document, and which permissions that apply.

This section describes which permissions apply to a document.

Note: The ACL of the document doesn't apply for the owner of the document. The owner will always have all rights to the document.

Note: If a document doesn't have an ACL applied, it is available to everyone.

Permissions that apply to a document:

Property	Description
Read	Lets the user read the document (metadata and content)
Modify	Lets the user modify the document
Delete	Lets the user delete the document
Approver	Lets the user approve a document created by another user (requires additional approval setup)

Overview of Dimension Access

iKnowBase uses the concept of dimensions to organize content. Every dimension may have an ACL applied to it. This ACL decides user permissions to the dimension.

This section describes which permissions apply to a dimension.

Permissions that apply to a dimension:

Property	Description
Categorize	Lets the user categorize content (documents) with this dimension.
Protected	Lets the user navigate to the dimension (and see documents tagged with this dimension, given that the documents themselves allow this). Note: There are two important scenarios here: <ul style="list-style-type: none">• A user may not have Protected access to the dimension, but still have Read access to the documents. Then, the user may not navigate through the dimension hierarchy, but may still find the document through other access paths (other dimensions, search, etc).• A user may have Protected access to the dimension, but still not have Read access to the documents. Then, the user will not be able to access the document through any mechanism.

Overview of Form Access

iKnowBase uses forms to edit or create content (documents). Every form may have an ACL which decides user access to the form.

This section describes which permissions apply to a form.

Permissions that apply to a form:

Property	Description
Creatable	Lets the user have access to this form, and use it to create and update content.

Overview of Task Wizard Access

iKnowBase uses task wizards to perform tasks and thereby create and update content (documents). Every task wizard may have an ACL which decides user access to the task wizard.

This section describes which permissions apply to a task wizard.

Permissions that apply to a task wizard:

Property	Description
Creatable	Lets the user have access to this task wizard, and use it to perform tasks and create and update content.

Overview of Page Access

iKnowBase uses pages to present content (documents). Every page may have an ACL which decides user access to the page.

This section describes which permissions apply to a page.

Permissions that apply to a page:

Property	Description
Read	Lets the user access this page. Note: The user may have Read access to a page, but still not have access to objects and documents on that page. Then, the user will not be able to access those objects and document through the page.

Overview of Subsystem Access

iKnowBase uses subsystem to categorize components used in the iKnowBase portal. Every subsystem may have an ACL which decides user access to manage components with the given subsystem.

This section describes which permissions apply to a subsystem.

Permissions that apply to a subsystem:

Property	Description
Read	Lets the user have access to manage components categorized with this subsystem. Note: A user may have Read access to the component itself. Then, the user will have access to the component, but not to manage it.

21. Metadata Administration

Metadata administration is the task of managing the metadata in the iKnowBase system.