



Grant agreement n°287589

Acronym: Micro B3

Start date of project: 01/01/2012, funded for 48 month

Deliverable 5.11

Community Service Documentation

Version: 2.1

Circulated to: Authors and Coordinator

Approved by: Prof. Frank Oliver Glöckner, 05/01/2016

Expected Submission Date: 31/12/2015

Actual Submission Date: 08/01/2016

Lead Party for Deliverable: MPI Bremen

Mail: rkottman@mpi-bremen.de

Tel.: +49(0)4212028974

Public (PU)	X
Restricted to other programme participants (including the Commission Services) (PP)	
Restricted to a group specified by the consortium (including the Commission Services) (RE)	
Confidential, only for members of the consortium (including the Commission Services) (CO)	



The Micro B3 project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 287589 (Joint Call OCEAN.2011-2: Marine microbial diversity – new insights into marine ecosystems functioning and its biotechnological potential).

The Micro B3 project is solely responsible for this publication. It does not represent the opinion of the EU. The EU is not responsible for any use that might be made of data appearing herein.

Summary

The Micro B3 Information System has community services components focusing on interoperability, visualization, and user interfaces. Some software components build the technical infrastructure such as Chon CMS and related components for information security. They are the basis for other community services components and tools such as PubMap and Metagenomic Trait Analysis. The former is a tool for crowd-sourcing geo-referenced scientific literature and the latter for metagenomic analysis and visualization of complex ecological traits. ProX is an example for web-based visualization to ease the comprehension of high-dimensional large-scale data sets such as networks resulting from metagenomic analysis. Furthermore, components like the MexMapWidget not only enhance user interfaces to maps of high-dimensional data, but are also modularized on the client site such that it can be embedded in any third-party web-page. Behind each web page are web services for the exchange of data between MB3-IS and related resources in various commonly used data formats which let other systems operate with the Micro B3-IS.

Table of Content

Introduction.....	3
Chon CMS	4
Web Services	5
Information Security	6
Authentication.....	6
Authorisation.....	6
Oauth 1.0a for authorization of web services.....	7
PubMap	8
PubMap Curator.....	8
PubMap Browser.....	12
Metagenomic Trait Analysis Service	13
Processing.....	13
Web services	14
Web Pages.....	14
ProX	16
MegxMapWidget.....	17
Add layer	18
Re-arranging layers on the stack.....	18
References.....	19
Appendix.....	20
Chon CMS Guide.....	20
Micro B3-IS Web Platform Security.....	20

Introduction

From the very beginning starting with requirements gathering and first functional specification it became apparent that the Micro B3 Information System (Micro B3-IS) has to have a web based distributable, data-centric, modular, and extensible architecture and needs to be based on existing standards and software.

There is already a plethora of web based frameworks and content management systems available. All content management system focus on managing the creation, editing, and display of web pages. Many of which have own custom ways of being extensible. However, they do not support data-centric architectures which are allowing data gathering and delivery in other ways than HTTP and HTML based. And none of them support distributing the system across several hosts. Whereas many existing web based frameworks allow developing such systems from scratch using their low-level application programming interfaces, they have non-standard ways of being extensible and only poorly support a modular design.

Therefore, the major architectural decision was to use the OSGi standard as the basis for developing main parts of the Micro B3-IS. OSGi is an industry standard for developing modular applications that can be used in a diverse range of environments from small embedded systems to large-scale distributed systems. It is non-restrictive and still allows using a combination of several frameworks, whereas existing frameworks would exclude the possibility to still use third-party frameworks.

Utilizing OSGi allows to develop the Micro B3-IS as 3-Tier web application based on modular OSGi services as shown in Figure 1. This allows to not only build web applications as typically done by content management systems, but also web services utilizing the same base services as e.g. for accessing data from the underlying database.

Based on these several other components have been developed which focus on interoperability, visualization, and user interfaces. All these components are summarized here as community services.

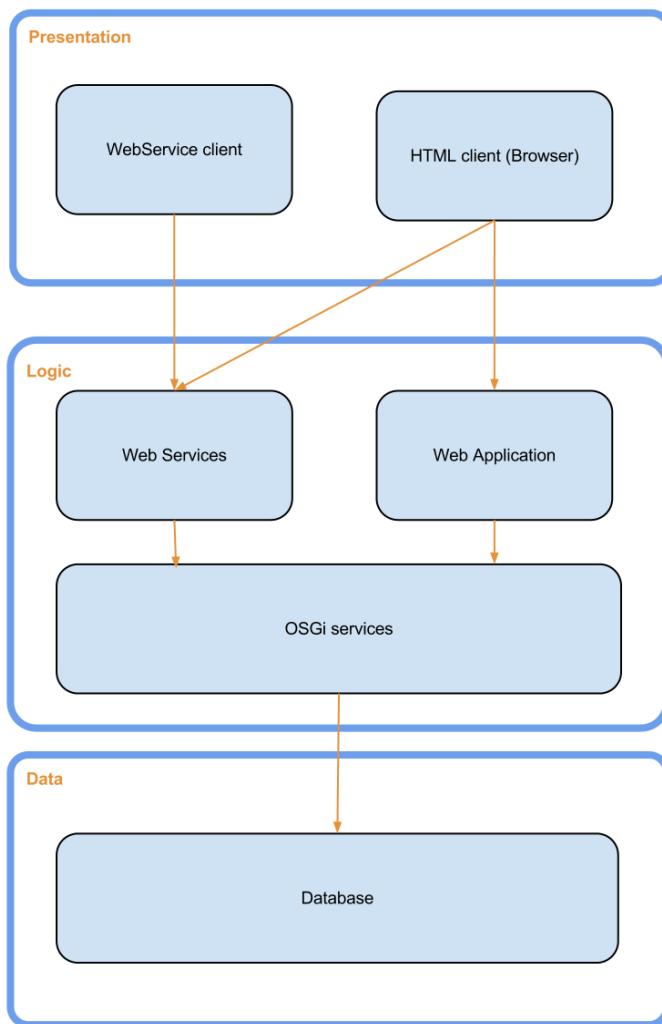


Figure 1: Three layer architecture of the Micro B3 Information System.

Chon CMS

Chon CMS is an OSGi based platform for data-centric web application development. By default it has plugins for simple web page content management. The content management is based on the Java Content Repository standard. Content is organised as trees and can be seen in simple terms as file like system.

Chon CMS implements a specific mechanism to e.g. bind a HTTP requests to certain content nodes of the content tree to then assemble and render the content as HTML in a web browser (see Figure 2).

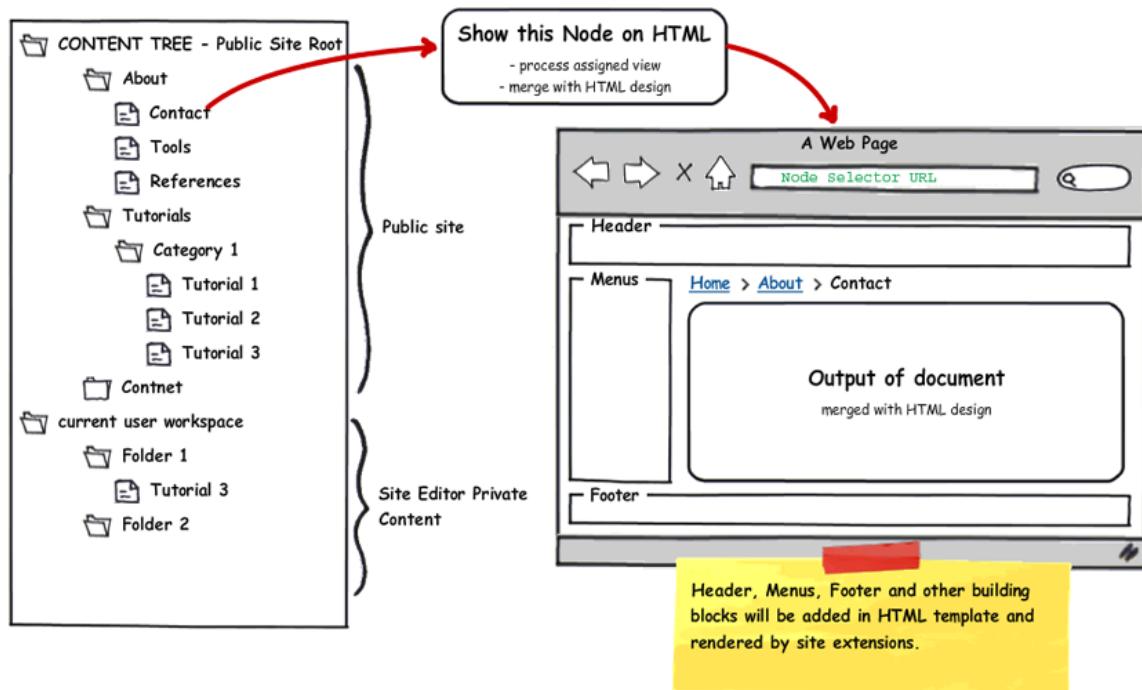


Figure 2: Simple binding of a request to content node “Contact” (left) and HTML generation for web browser (right). Courtesy of InterWorks.

With this concept, data (content) is separated from presentation. Administrators can create content that will have data only, not worrying about how that content will be presented.

Web Designers have an easy access to all system variables and content properties. Each content node can have its own template for presentation. Complete look and feel can be changed without modifying content data.

Utilizing the OSGi standard for developing Chon CMS plugins, the system is a robust basis base for implementing any kind of WCM feature e.g. for automated templates, access control, workflow management, collaboration, delegation, document management among others. More documentation is available in [Chon CMS guide](#).

Web Services

Web services are services mainly used for interoperable machine-to-machine communication. Consequently, numerous web services were developed in the context of the Micro B3-IS.

Several new services were developed by existing infrastructures like European Nucleotide Archive, SeaDataNet, EurOBIS and were already described in several deliverables.

Most of the other new web services were developed in the context of Ocean Sampling Day and are hosted on megx.net (<http://mb3is.megx.net>). These are implemented following the RESTful architectural style [1], [2], [3].

Information Security

Although most web services and web sites can be used without authentication and authorisation, there are few cases where such mechanisms are required in order to achieve a trust relationship with the user. Standardized mechanisms were implemented for authentication and authorization of access to both web pages and web services. Technical details are documented in the Appendix "[Micro B3-IS Web Platform Security](#)".

Authentication

One major use case requiring authentication is the upload of data from the OSD Smartphone App. Here the user first needs to register either a new account or login via the user's Facebook and Twitter account (see Figure 3). Once an account is registered the user can then login from the OSD App which then authorizes the app to upload OSD data to the OSD Registry web service.

The screenshot shows a login page titled "Login with your Megx credentials:". It features two input fields: "Username" with a user icon and "Password" with a lock icon. Below these is a blue "Login" button. To the right of the password field, the text "You can also" is followed by two buttons: "Connect with Facebook" (with a Facebook icon) and "Sign in with Twitter" (with a Twitter icon). At the bottom, a message says "You don't have an account yet?" with a link "Create it here".

Figure 3: Screenshot of the login/register web site of [megx.net](#) (new layout not publically available yet).

Authorisation

MB3-IS implements a classical role based access control (RBAC) mechanism. For web pages it means that knowing who the user is (after an authentication procedure, see above), the access to certain pages can be completely denied or certain functions or content of a web page is only enabled if the user has the right role. For example access to the Chon-CMS page for editing web content online e.g. <http://mb3is.megx.net/admin/content.edit.do?path=/www> is only allowed to users with the role 'admin'. In case the user isn't authenticated the request will be redirected to the login page (<http://mb3is.megx.net/login>). If the user is authenticated but does not have the permission because of not being member of the role 'admin', the user gets a permission denied message. Otherwise the user gets access to the page as shown in Figure 4. In

Additional Figure 5 shows a screenshot of a MB3-IS web page where the user 'megx' is authenticated and therefore a link to directly edit the content of the page is shown on the top-left of the page.

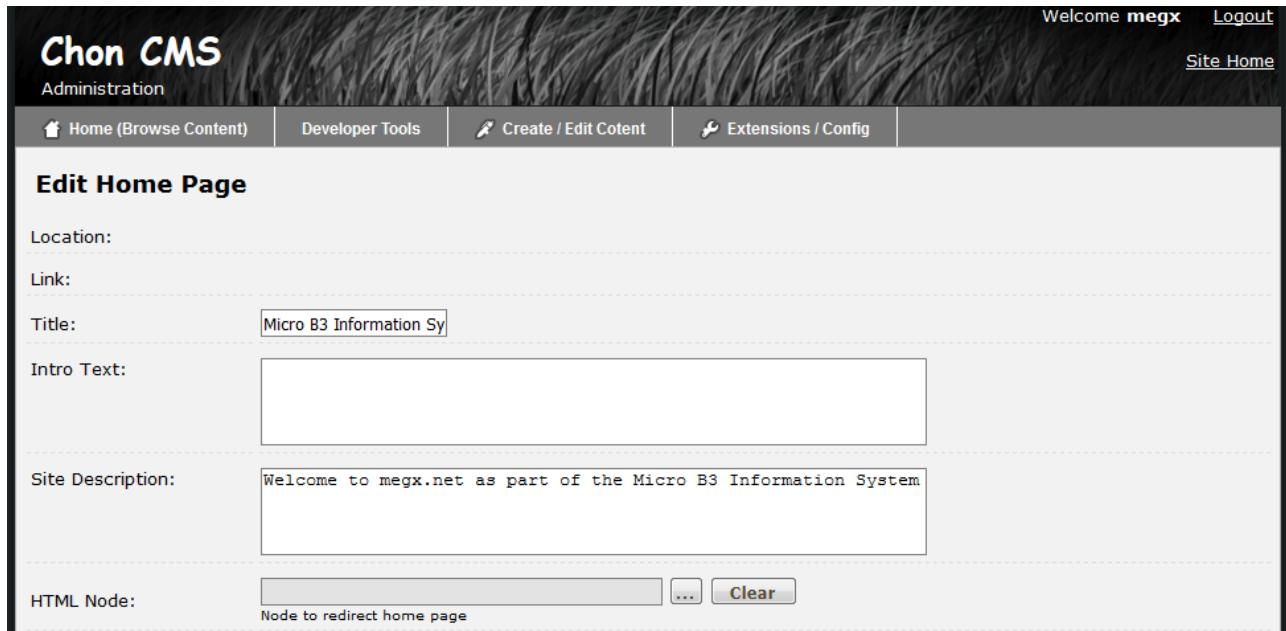


Figure 4: Chon-CMS web content edit page. Only users with role 'admin' are authorized to access this web page.



Figure 5: Special content is only shown if user is authorized. Here, the user 'megx' is in role admin and therefore a link is displayed on the top-left to edit the content of this page.

Oauth 1.0a for authorization of web services

The above described authentication and authorisation mechanism does not fully work in the context of web services for the machine-to-machine communication because they require a human to perform the login in order to access restricted pages. Therefore MB3-IS implements OAuth1.0a. It is an open protocol allowing secure authorization in a simple and standard method from web, mobile and desktop applications. The end user's information is securely transferred without revealing the identity of the user. In recent years OAuth became the de-facto standard in the Internet industry and is supported by all major Internet companies like Google, Twitter, and Facebook among others. This way, a user can then login from the OSD App which then authorizes the app to upload OSD data to the OSD Registry via a secured web service.

PubMap

The World Map of Publications (PubMap) system was developed to crowdsource the creation of a georeferenced bibliography (see Figure 6). This includes locations worldwide and it is not limited to certain journals or fields of study. PubMap has two main components, the PubMap Curator (PMC) a bookmarklet for manual georeferenced annotation of publications and the PubMap Browser (PMB) a web application to retrieve those georeferenced publications. Each publication should be annotated depending on the origin of the study material not on the location of the university or institute at which the study was conducted. For example, if a publication discusses a water sample which was taken in the Pacific Ocean but the measurements and analyses were done at Harvard University, the publication should be georeferenced with the location of the Pacific Ocean and not of Harvard. The idea is to create a database which can be filled with information by volunteers and the stored information is openly available and can be retrieved by everyone. PubMap is based on Chon CMS and hence integrated in the Micro B3 Information System (Micro B3-IS).

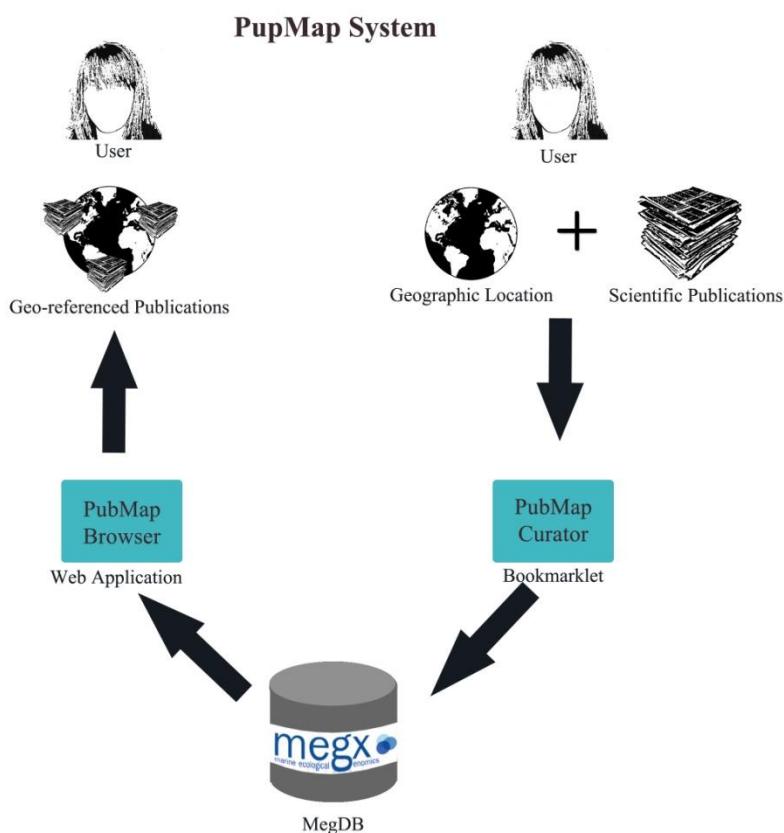


Figure 6: The user reads a scientific publication and searches for the geographic location of the sampling locality in the publication text. The user enters this information into the PubMap Curator bookmarklet. The data is then stored in the MegDB database. The georeferenced publication can then be retrieved via the PubMap Browser. The user can search for publications of a specific geographic location on a world map or via text search in a table.

PubMap Curator

Users who want to annotate publications with geographic coordinates can use the bookmarklet PubMap Curator. A bookmarklet is a small program coded in JavaScript which is saved in the bookmark section of a web browser such as Firefox or Chrome. The PMC bookmarklet will be made available at <http://mb3is.megx.net/pubmap/curation>¹ and is installed by dragging and dropping it into the bookmark bar of the browser (Figure 7). When PMC is started, the user needs to first log in via a megx.net's Facebook or Twitter account. Currently, automatic retrieval of article metadata only works on abstract sites of NCBI's PubMed library. The user can choose three different options for the geo-referenced annotation. He can either 1. enter the latitude and longitude in decimal degrees or 2. in degrees, minutes and seconds. To avoid mistakes users will get a warning if they enter coordinates which are out of the existing range. 3. If the geographic coordinates are unknown, the user can choose a country or sea name from a provided list and then manually enter the place name. When the user clicks on the save button the information is send to the server and stored in the Microbial Ecological Genomics Database (MegDB) the backbone of megx.net and the Micro B3-IS (see Figure 8). To keep the workload at a minimum the bookmarklet was designed to be used as straight forward as possible; hence, the average time to fill out the required information is about 30 seconds. To ensure consistent naming of locations another web service takes the entered information and searches, either with the given geographic coordinates for a place name with the GeoName web service "search" (<http://www.geonames.org/export/geonames-search.html>), or uses the place name to retrieve the geographic coordinates from the GeoNames web service "extendedFindNearby" (<http://www.geonames.org/export/web-services.html#extendedFindNearby>) (see Figure 9). All data submitted with the PMC can then be retrieved via the PubMap Browser. In the case a publication was already georeferenced beforehand; a message appears and informs the user. It also displays the geographic coordinates and the place name the publication is annotated with. That way the user can review whether the geographic information is correct. If he finds a mistake he can report it directly. If he wants to add additional geographic information he can fill in the fields of PMC and save it.

¹ A beta version is available at <http://iwgate.poweredbyclear.com:9080/megx.net/pubmap/curation>

Home Info OSD Tools Browse Edutainment Portal Contact Log in

PubMap Curator

How it works

1. Drag the "PubMap Curator" button to your Bookmarks Bar 
2. Find an article
Search for article on the  PubMed - NCBI site.
3. Click the "PubMap Curator" bookmark
Click the "PubMap Curator" bookmark you just created, and the article details will display on the right hand side of the page.
4. Review the details and Save
If the details look ok, please next insert the Geographic location of the study site then click the "Save" button.
Any newly saved articles will appear in the Pubmap Articles site.



A screenshot of a web browser window titled "A microfluidic system for stud...". The address bar shows "www.ncbi.nlm.nih.gov/p...". A blue button labeled "PubMap Curator" is visible in the bottom right corner of the window. Below the browser window, there is descriptive text for each step of the process.

Figure 7: The PubMap Curator website with detailed instructions on how to save the bookmarklet in the bookmarks bar and how to use it.

The figure consists of five vertically stacked screenshots of a web-based annotation tool, likely a bookmarklet for PubMed. Each screenshot shows a different step or state of the annotation process:

- Screenshot 1:** Shows the initial annotation page for a specific publication. It includes fields for entering geographic coordinates (Latitude and Longitude with decimal degree and DMS options), a dropdown for 'Select Country/Sea', and a text input for 'Enter Place Name'. A note at the bottom states: '(Please use the coordinates of the study site the article relates to and not the location of the university. For further information please visit [Help section](#))'. A 'Save' button is visible.
- Screenshot 2:** Shows the same interface but with the 'Latitude' field set to 'N+' and the 'Longitude' field set to 'E+', indicating coordinates in degrees, minutes, and seconds.
- Screenshot 3:** Shows the interface with the 'Select Country/Sea' dropdown open and the 'Enter Place Name' field populated with a place name.
- Screenshot 4:** Shows a green success message box stating 'Article was successfully saved!'.
- Screenshot 5:** Shows a message box indicating that the article is already in the database curated with the location. It provides coordinates (Lat: 15.309548, Lon: -74.676078) and offers options to report wrong geo-references or contact support.

Figure 8: Screenshots of the PubMap Curator bookmarklet with the three different options to enter either geographic coordinates or a place name. 1. Shows the option to enter geographic coordinates in decimal degree and 2. in degrees, minutes and seconds. 3. Shows the text fields which should be filled in if the coordinates are unknown. 4. The message displayed after successful saving of data. 5. The message displayed in case the publication was already annotated including the button to report wrong annotations.

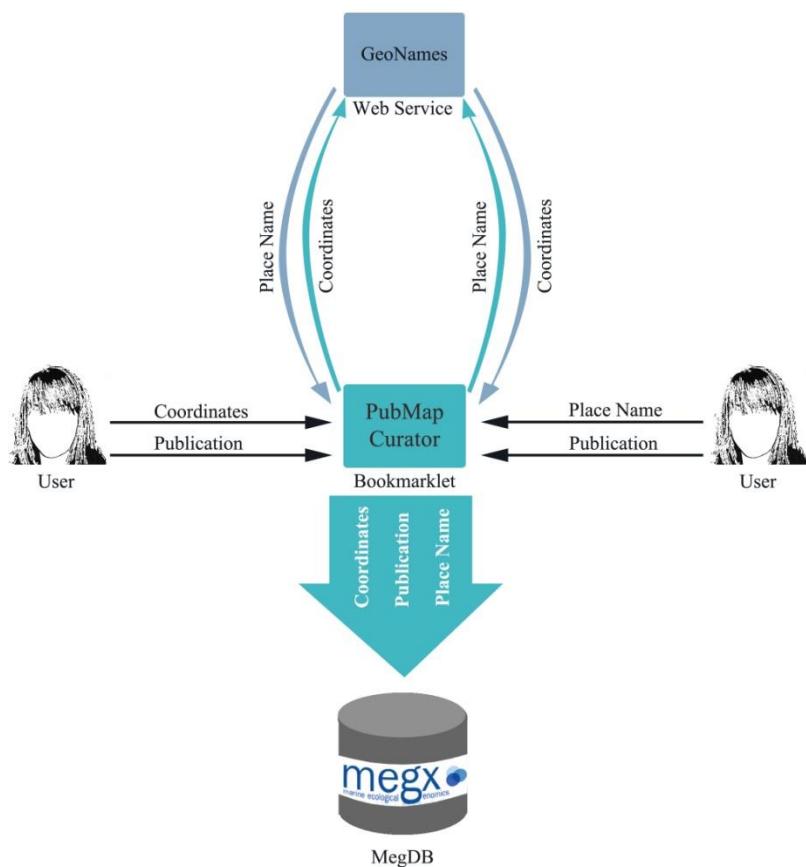


Figure 9: Utilization of the GeoNames web services. If the user enters the contextual data of a publication together with geographic coordinates into the PubMap Curator the coordinates will be send to the GeoNames web service and a place name will be retrieved (left). If the user enters a place name instead of coordinates, the place name will be send to GeoNames web service and geographic coordinates will be retrieved (right). Subsequently, coordinates, publication data and the place name are all saved in MegDB.

PubMap Browser

The PubMap Browser will be made available at <http://mb3is.megx.net/pubmap/list>². It is a web application giving users the possibility to search for scientific publications which study specific geographic locations (see Figure 10). It contains a world map showing all georeferenced publications stored in the database at that point of time. The publications are visualized as coloured dots on the world map—according to the geographic coordinates the publications are annotated with—on the location where the study took place or the study material was coming from. The world map is dynamically created using the Genes Mapserver component. Underneath the world map all georeferenced publications are represented in form of a table. The table shows the title, authors, journal, publication date, world region and place name of the publication, as well as a direct link to the NCBI PubMed abstract website. Additionally, the user has the option to use text based search on the table for e.g. a specific location, topic or author name.

² A beta version is available at <http://iwgate.poweredbyclear.com:9080/megx.net/pubmap/curation>

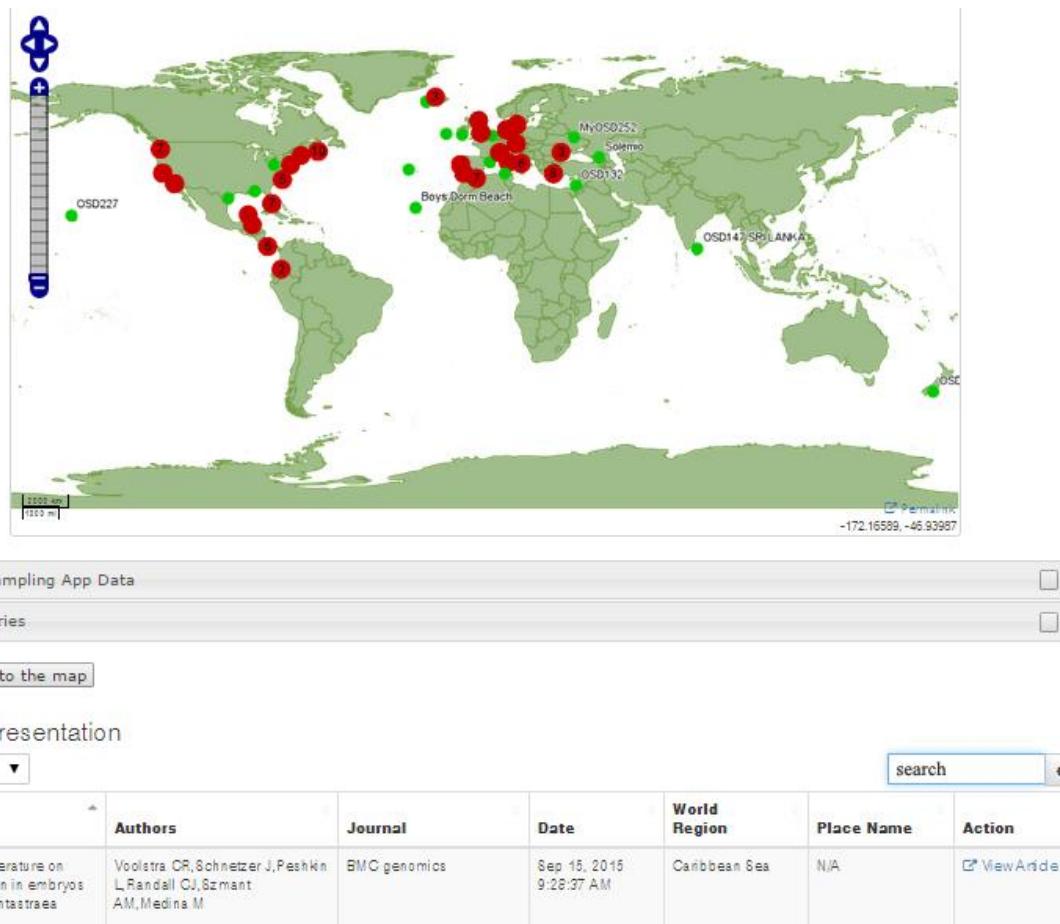


Figure 10: PubMap Browser displaying the georeferenced publications stored in the database. The colored dots on the world map represent publications. The table holds all publications and is text searchable.

Metagenomic Trait Analysis Service

The Microbial Metagenomic Traits Analyses is a set of components for calculating a set of metagenomic traits described in Barberan et al. 2012 "Exploration of community traits as ecological markers in microbial metagenomes"[4] for a large variety of public, microbial metagenomes. The traits included in the statistical analyses range from GC content to functional diversity, and deliver a valuable set of ecological markers in order to discriminate between habitats, geographic locations, or temporal samples. Furthermore, inter-trait relationships can be used as habitat descriptors or indicators of artefacts during sample processing.

As such the metagenomic trait analysis goes beyond classical bioinformatics analysis for metagenomes and analysis these within a well-defined ecological framework based on traits analysis.

Processing

The calculation of traits is done in a complex pipeline mostly implemented using Bash and R scripts distributed on a compute cluster. All results are written into files and imported into MegDb on successful completion.

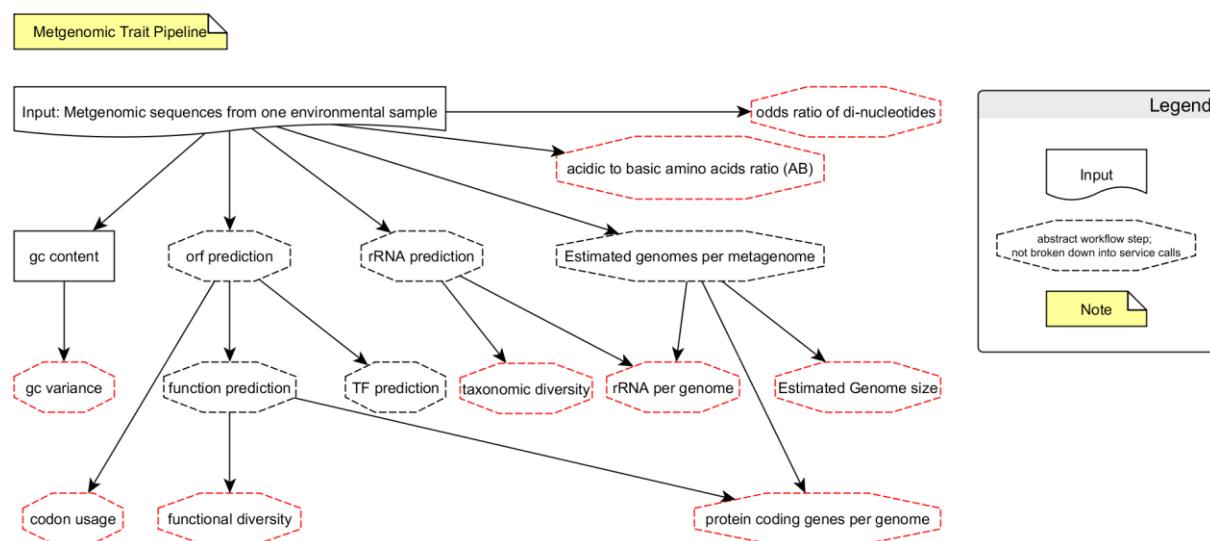


Figure 11: Schematic view on what trait gets computed based on a metagenome data set from a single environmental sample.

Web services

There are corresponding web services for each metagenomic trait. These services were developed within in the BioVeL EU Project and are well documented on the Biodiversity Catalogue at <https://www.biodiversitycatalogue.org/services/64>. Nevertheless, they are hosted on megx.net (<http://mb3is.megx.net>) and follow the RESTful architectural style of the Micro B3-IS.

Web Pages

Currently, a user can get list of all pre-calculated traits at <http://mb3is.megx.net/mg-traits/samples> (Figure 12). Currently, all traits are calculated from the Global Ocean Survey (GOS) and Ocean Sampling Day 2014 metagenomes. The tabular view allows a user to download all data for offline use or interactively search for metagenomes of interest. Details of all traits for each metagenomes can then be browsed by following the “View more” link in the most right column. For example for the metagenome OSD 3 the page (<http://mb3is.megx.net/mg-traits/sampleDetails?id=902>) gives details of all traits of this metagenome including interactive charts for more complex traits like e.g. “Aminoacid Trait” (Figure 13).

The user can also explore the data at <http://mb3is.megx.net/mg-traits/traits-summary> from the perspective of how the samples cluster on traits like amino acid content, codon usage, di-nucleotide odds ratio, gene functions, and taxonomy (Figure 14).

Download All

	Sample Label	Sample Name	Environment	Size	Number of sequences	Actions
OSD1	OSD1_2014-06-21_0m_NPL022	marine biome	255244805	937467	View more	
OSD3	OSD3_2014-06-20_0m_NPL022	marine biome	239538946	833614	View more	
OSD5-75m-depth	OSD5_2014-06-19_75m_NPL022	marine biome	236127334	766231	View more	
OSD6	OSD6_2014-06-21_0m_NPL022	marine biome	248559294	775101	View more	
OSD5-1m-depth	OSD5_2014-06-19_1m_NPL022	marine biome	238695244	835572	View more	
OSD4	OSD4_2014-06-20_0m_NPL022	marine biome	209560966	976344	View more	
OSD15-50m-depth	OSD15_2014-06-21_50m_NPL022	marine biome	255575682	851430	View more	
OSD17	OSD17_2014-06-20_3m_NPL022	marine biome	241168962	772492	View more	
OSD22	OSD22_2014-06-23_1m_NPL022	marine biome	198680592	564456	View more	
OSD36	OSD36_2014-06-21_0.5m_NPL022	marine biome	350560149	1286261	View more	

Showing 1 to 10 of 218 entries [Download selected data](#)

First Previous 1 2 3 4 5 ... 22 Next Last

Figure 12: Tabular overview of all pre-calculated traits with options to sort, search and download data (<http://mb3is.megx.net/mg-traits/samples>).

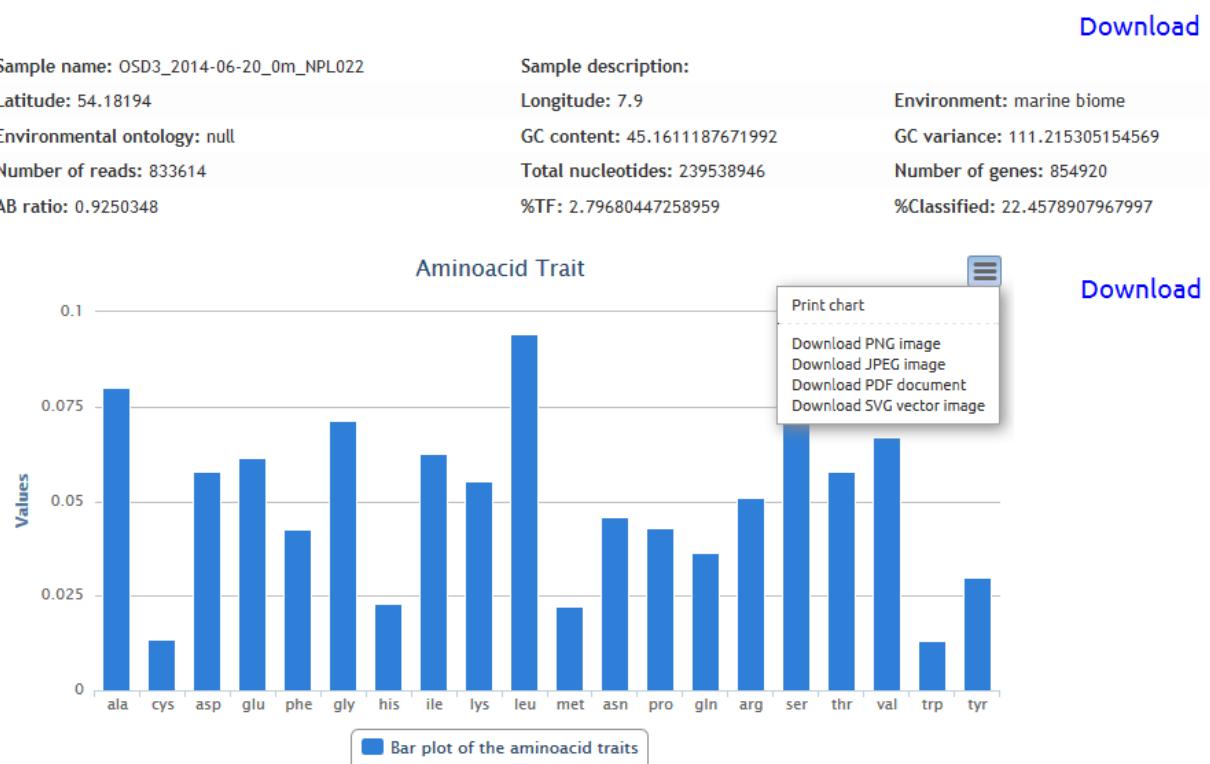


Figure 13: Web page showing all traits of the metagenome from OSD3. At the top is a table showing values of simple traits i.e. having just a single value. This is followed by several interactive charts for more complex traits. Shown is the “Aminoacid Trait”. Others are not shown on screenshot but can be seen at <http://mb3is.megx.net/mg-traits/sampleDetails?id=902>. Each chart can be downloaded in different formats for use in e.g. in publications.

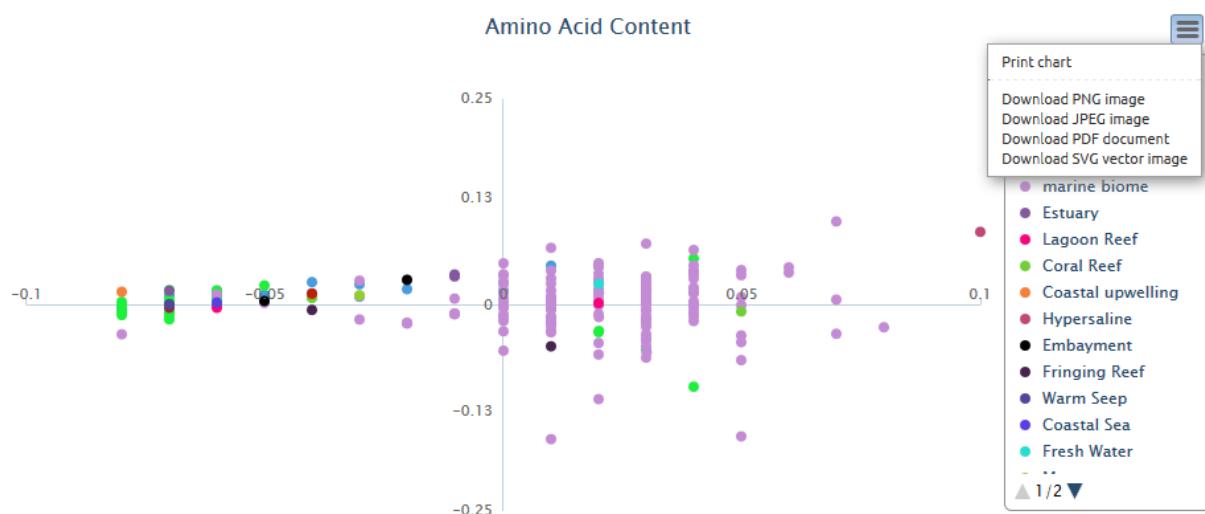


Figure 14: View on all metagenomes clustered by traits. Shown is cluster by “Amino Acid Content” others can be browsed at <http://mb3is.megx.net/mg-trait/traits-summary>. Colours indicate environment of the sample. Each chart can be downloaded in different formats for use in e.g. in publications.

ProX

In collaboration with the University of Applied Sciences, Bremen Matthias Stock developed the Protein Explorer (ProX application)[5]. It is entirely written in JavaScript and hosted on megx.net as Chon CMS plugin. Although ProX can be used for any kind of graph, its main purpose is to allow all users to explore known-unknown networks based on PFAMs. Performance measurements and comparisons to other products proofed that no library for exists that is neither powerful enough to render the given graph from in a web browser on commodity hardware nor as feature-complete as comparable applications. It has a built-in node search functionality based on the PFAM name or PFAM accession; it also retrieves metadata from each node from the PFAM web services and the user can create ego networks centered to their desired node (Figure 15).

Currently, ProX is only available with special permission (see Authorisation) which is given on demand. A short video demonstrates the functionality of ProX and can be watched here <https://www.dropbox.com/s/8ztqxtqhir726nu/prox.mp4?dl=0>.

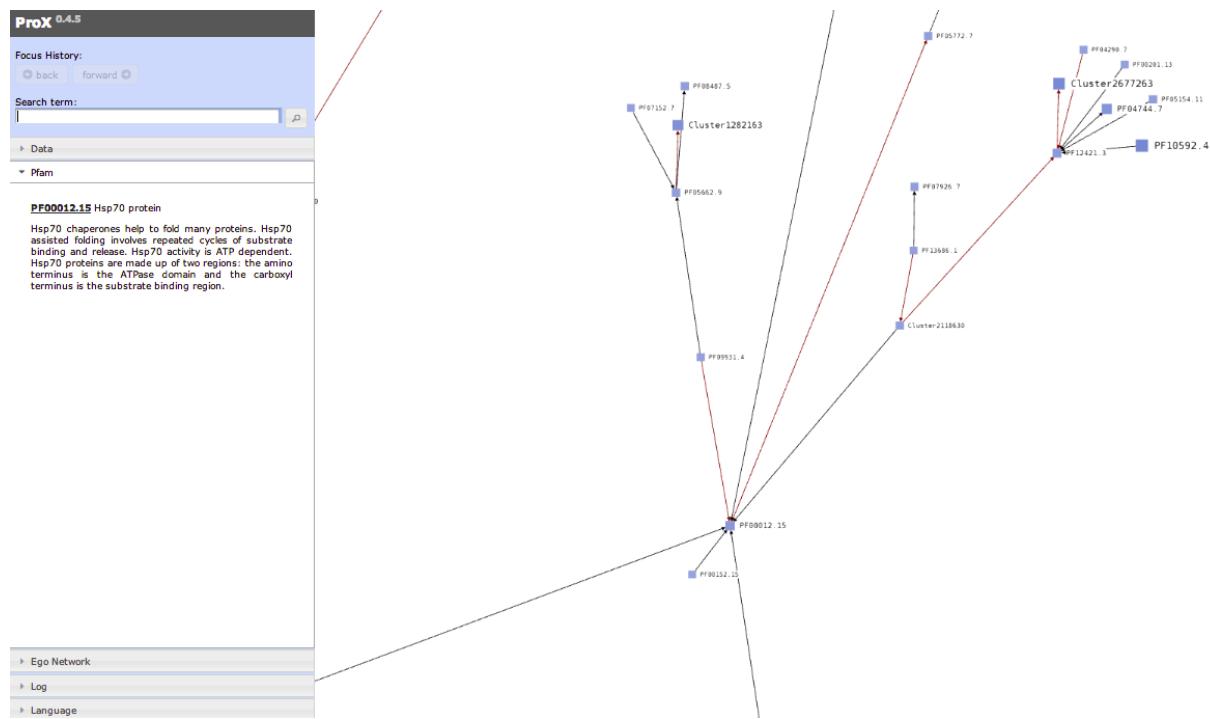


Figure 15: Screenshot of the web based Protein Network (ProX) application. Left, application panel incl. search box and details on a selected PFAM among other functionalities. Right, view on an ego-network.

MegxMapWidget

A widget is a small stand-alone application that can be installed and executed within third-party web pages. The MegxMapWidget retrieves data from the Genes Mapserver and displays maps alongside some handles to interactively organize the different map layers dynamically.

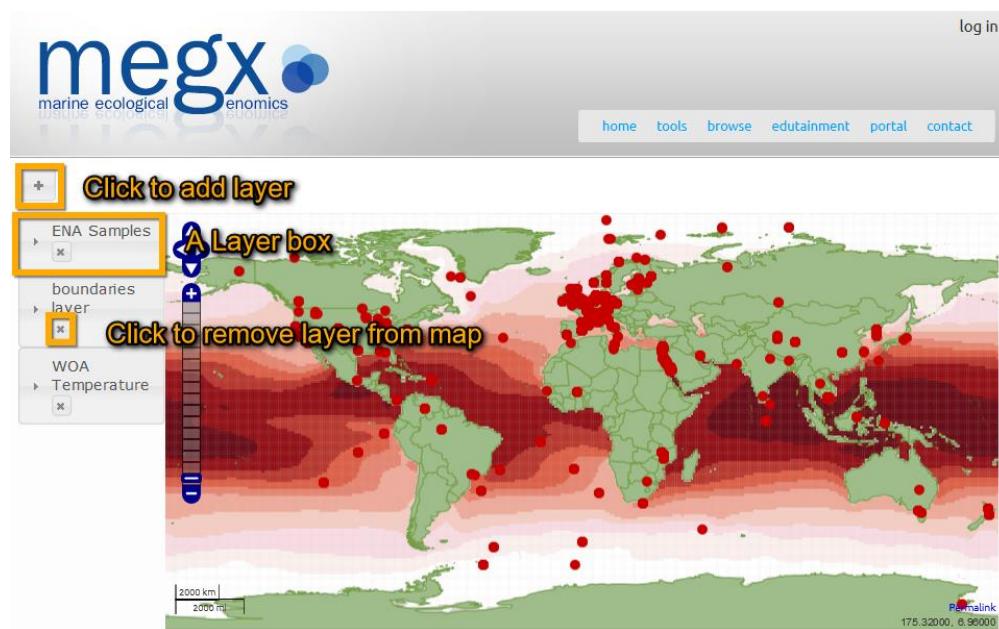


Figure 16: Screenshot of MegxMapWidget with handles to manage layer display alongside typical handles for zooming, panning and clicking.

Adding layers

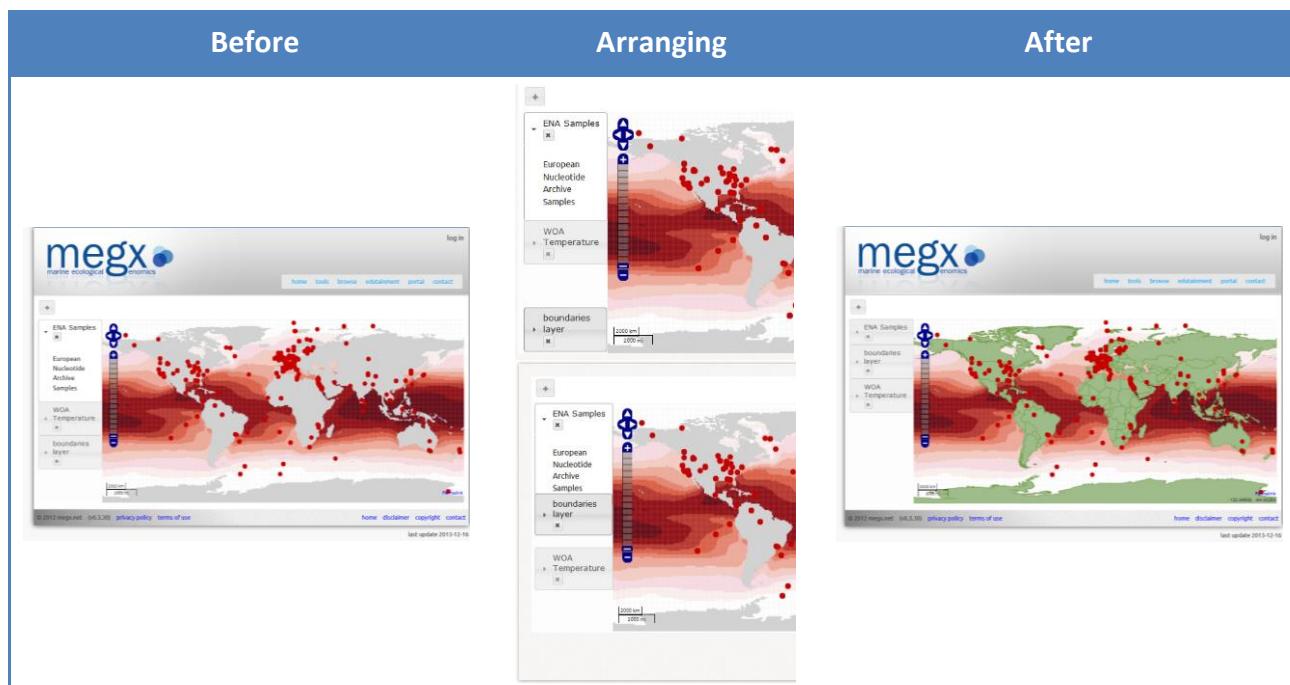
Click on the '+' button, then the dialog as shown below will appear and you can choose the desired layer.



Figure 17: MegxMapWidget showing add layer dialog.

Re-arranging layers

Just left-click and hold a layer box and move up in order to render the chosen layer higher on the top of the map. Below is a sequence of screenshots demonstrating the effect of moving the 'Boundary layer' on top of 'WOA 05 temperature' layer (but still below 'ENA Samples').



References

- [1] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, 2000.
- [2] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, Inc., 2007.
- [3] L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs*. Sebastopol, California: O'Reilly, 2013.
- [4] A. Barberán, A. Fernández-Guerra, B. J. M. Bohannan, and E. O. Casamayor, "Exploration of community traits as ecological markers in microbial metagenomes," *Mol. Ecol.*, Nov. 2011.
- [5] M. Stock, "Efficient Web-Based Visualization of Complex Data Sets in Marine Microbiology," Hochschule Bremen, 2013.

Appendix

[Chon CMS Guide](#)

[Micro B3-IS Web Platform Security](#)

InterWorks

Chon CMS Guide



The Micro B3 project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 287589 (Joint Call OCEAN.2011-2: Marine microbial diversity – new insights into marine ecosystems functioning and its biotechnological potential).

The Micro B3 project is solely responsible for this publication. It does not represent the opinion of the EU. The EU is not responsible for any use that might be made of data appearing herein.

Contents

<u>1</u>	<u>Introduction</u>	24
<u>1.1</u>	<u>What is Chon CMS</u>	24
<u>1.2</u>	<u>How it works</u>	24
<u>1.3</u>	<u>The content tree concept</u>	26
<u>1.3.1</u>	<u>Node Types</u>	26
<u>1.3.2</u>	<u>Public Node – “/www”</u>	26
<u>1.3.3</u>	<u>Category Node Type – “category”</u>	26
<u>1.3.4</u>	<u>HTML Content Node Type – “html”</u>	27
<u>1.4</u>	<u>OSGi plugin concept</u>	27
<u>2</u>	<u>Chon CMS development</u>	28
<u>2.1</u>	<u>Installation</u>	28
<u>2.1.1</u>	<u>Install Maven</u>	28
<u>2.2</u>	<u>Generate Chon CMS Project Structure</u>	28
<u>2.3</u>	<u>Build</u>	28
<u>2.4</u>	<u>Run</u>	29
<u>2.4.1</u>	<u>Run on Jetty server</u>	29
<u>2.4.2</u>	<u>Run on any application server as war</u>	30
<u>2.5</u>	<u>Template development</u>	30
<u>2.6</u>	<u>Creating new plugin</u>	30
<u>2.7</u>	<u>Setup Eclipse environment</u>	33
<u>2.7.1</u>	<u>Setting OSGi Target Platform</u>	33
<u>2.7.2</u>	<u>Importing Plug-in Projects</u>	35
<u>2.7.3</u>	<u>Build Plugins from eclipse</u>	35
<u>2.7.4</u>	<u>Importing Web Application and configure tomcat web server</u>	37
<u>2.8</u>	<u>Plugin Development</u>	39
<u>2.8.1</u>	<u>Activator</u>	39
<u>2.8.2</u>	<u>Extensions</u>	40
<u>2.8.3</u>	<u>Node Types</u>	40
<u>2.8.4</u>	<u>Repository setup file</u>	40
<u>2.8.5</u>	<u>Configuration options</u>	40
<u>2.9</u>	<u>Content Model</u>	40
<u>2.9.1</u>	<u>Initial repository structure</u>	41

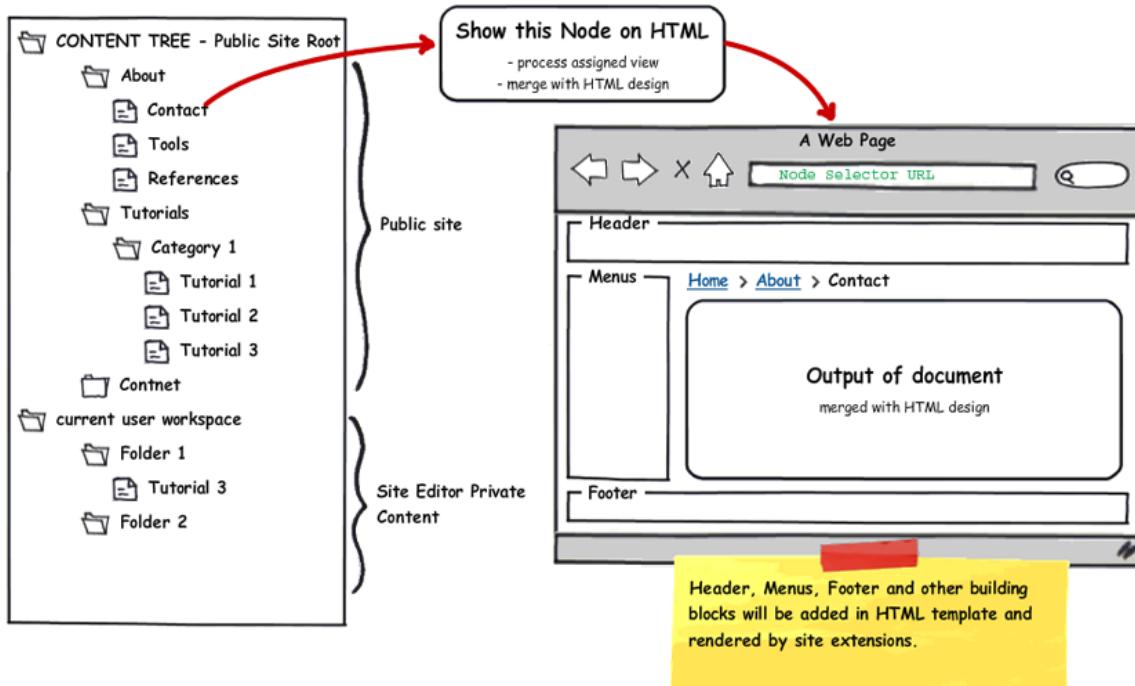
2.9.2	Retrieve content node	42
3	Existing Plugins	43
3.1	Third Party Plugins	43
3.1.1	Felix Framework – org.apache.felix.framework	43
3.1.2	OSGi services – osgi.comendum	43
3.1.3	Apache HTTP services – org.apache.http.*	43
3.1.4	Apache Felix Fileinstall – org.apache.felix.fileinstall	43
3.2	Core Plugins	43
3.2.1	Legacy jars packed in OSGi bundle – org.choncms.bnd-libs	44
3.2.2	Utils - org.chon.core.utils	44
3.2.3	Commons - org.chon.core.common	44
3.2.4	Web - org.chon.web	44
3.2.5	Core - org.chon.cms.core	44
3.2.6	Model - org.chon.cms.model	44
3.2.7	JCR Client - org.chon.jcr.client	45
3.3	Administration Plugins	45
3.3.1	Admin - org.chon.cms.admin	45
3.3.2	Editor - org.chon.cms.admin.editor	45
3.3.3	Explorer - org.chon.cms.admin.explorer	45
3.4	UI Plugins	45
3.4.1	Content Utils - org.chon.cms.content	45
3.4.2	Display Lists - org.choncms.display.lists	45
3.4.3	jQuery - org.chon.cms.ui.jquery	45
3.4.4	Lightbox - org.chon.cms.ui.lightbox	45
4	Technologies used in Chon CMS	46
4.1	OSGi	46
4.2	Java Content Repository (JCR)	46
4.3	Apache Felix	46
4.4	Apache Velocity Template	46
4.5	Eclipse PDE	47
4.6	Tycho Maven Build	47

INTRODUCTION

What is Chon CMS

Chon CMS is a plugin based platform for content based web application development. Chon CMS itself is highly modularized and OSGi based. By default it comes with plugins for web page creation, administration and simple content management backend. The content management is based on content trees and can be seen in simple terms as file like system based on the Java Content Repository standard.

Chon CMS implements a specific mechanism to e.g. bind a HTTP requests to certain content nodes of the content tree to then assemble and render the content as HTML in a web browser (see [Figure 18](#)).



[Figure 18: Simple binding of a request to content node "Contact" \(left\) and HTML generation for web browser \(right\)](#)

With this concept, data (content) is separated from presentation. Administrators can create content that will have data only, not worrying about how that content will be presented. Content is saved in java standard repository that physically can be persisted in file system, any JDBC database, cluster or cloud.

Designer will have easy access to all system variables and content properties. Each content node can have its own template for presentation, or node template can be selected based on node type. Complete look and feel can be changed without modifying content data.

With OSGi plugins flexible extension, Chon CMS is ready base for implementation of any WCM feature: automated templates, access control, workflow management, collaboration, delegation, document management, version etc...

How it works

Chon CMS in its core deployment is simple web application that serves as OSGi plugin container. It uses apache Felix for running the OSGi and dispatches all requests to bundles that have request service listeners. Chon CMS comes with few plugins that serve and manage simple content; core plugin have one service listener that process the request based on resource type (see [Figure 2](#))

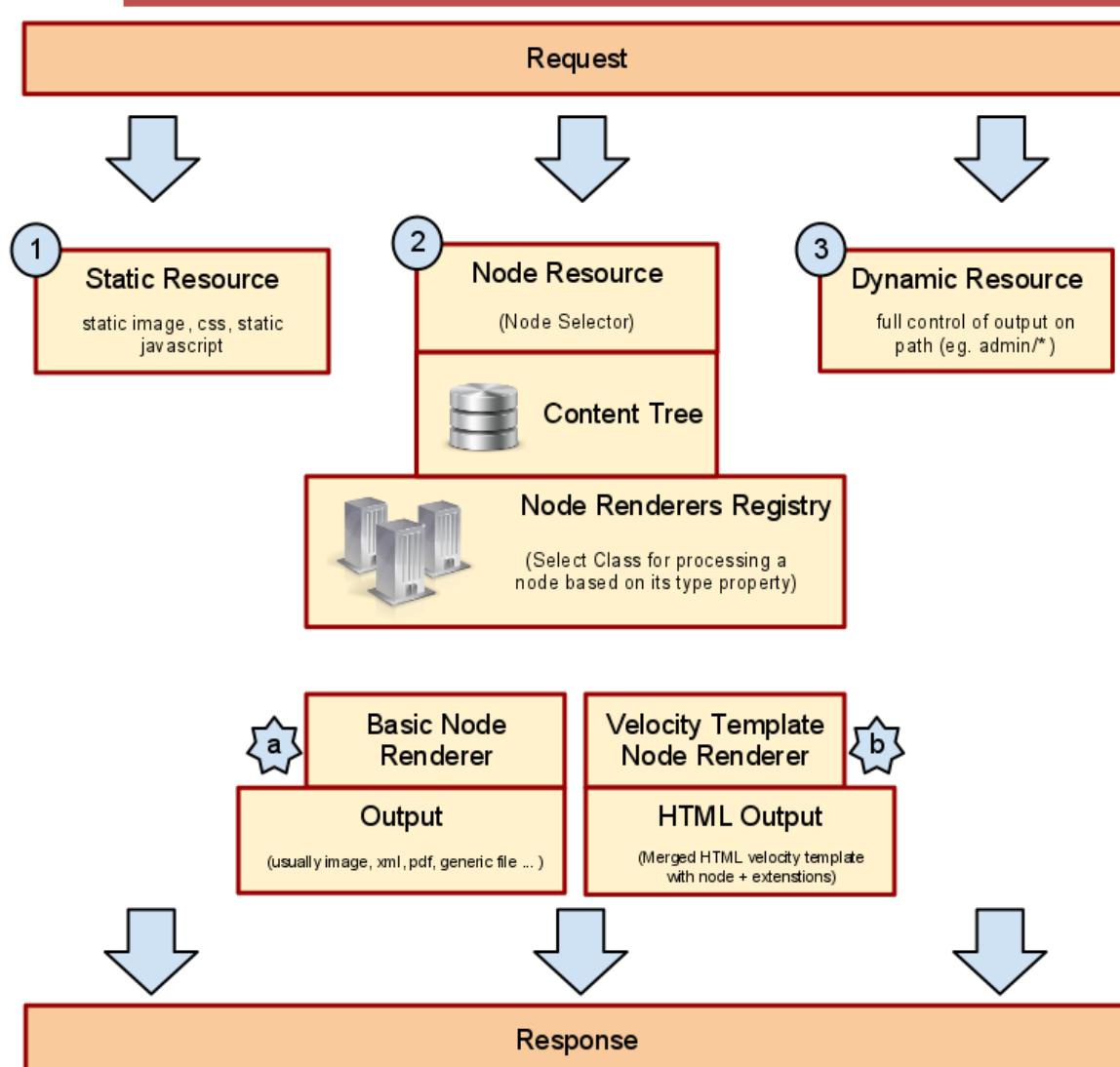


Figure 2: Diagram for request/response pipeline, center layer (node processing) is the most important part

Each request to web application can go through one of 3 braches for processing, depending on existing resource:

- 1) **Static Resource** - static files deployed in res folder in each bundle. All "res" folder are automatically exposed to server static resources. When server finds static file (specified by request), it returns file contents. Example for such request would be http://<SITE_URL>/images/logo.png
- 2) **Node Resource** – content node. When request is not for static file, application will check repository for (semi-dynamic) content. If node is found that node is sent for further processing:
 - Basic Node Renderer** – Instance of INodeRenderer that can output a node in whatever (programmed) format. Developer will register instance of INodeRenderers in OSGi context if logic for an application require specific node processing. Example would be RSS. Developer would register node type "rss" that will have RSSNodeRenderers registered in OSGi context that will render xml file for latest content.
 - Velocity Template Node Renderer** – Instance of INodeRenderer that is internally implemented to output velocity template. VtplOutNodeRenderer from org.chon.cms.core bundle is the name of class implementing this logic. Instance of this class is registered in OSGi registry with process node request, puts \$this (current node) along with other global templating variables in context, and renders final output using velocity.
- 3) **Dynamic Resource** – special resource type with full control of output. Usually this type will not be used; this is internally used in admin part of Chon CMS.

The content tree concept

The content tree model is based on the Java Content Repository (JCR) standards as implemented by Apache Jackrabbit. In Chon CMS each node has a property “type” which determines the processing units to be applied before rendering- and which renderer to use for the node content. In “Figure 2”, scenario 2 is showing content node processing.

Chon CMS repository have one public folder from where nodes are selected for public presentation. That folder is “/www” with type: sys.public.

Node Types

Node type determinates how that node will be shown on output. For example, file node type will show binary data while html node type will show HTML content merged with velocity template. **OSGi service registry selects node renderer based on node type.**

By default Chon CMS has 3 main types of nodes:

Home Page (www node)

- type: sys.public
- template: pages/home.html

Category Node

- type: category
- template: pages/category.html

HTML Content Node

- type: html
- template: pages/view.html

All default types will render HTML using velocity template. Initially base.html template is selected (as generic template for all velocity template node types) and based on node type sub template is selected for node type specific rendering content. For example, a category would list all its children (in category.html), but html content node would render its data. Home Page would either render content from extensions (news, widgets) in home.html or can be redirected to other content node.

There is another type of base node – a file node. Each uploaded file (type: nt:file) will be rendered by simple output file binary data. Server can detect type of file and process additional rendering parameters to file, for example if image file is requested, server can process “w” and “h” parameters to output image with specific dimensions (width and height).

Types and their default renderers are defined in /etc/types and initially are created using setup.kson.

Public Node – “/www”

Only one root node is exposed for anonymous user – “/www” node (sys.public). Children of this node are directly accessible directly on request. For example if request is http://<SITE_URL>/my.node.html, system will search for /www/my.node.html and if found it will be returned (processed by INodeRenderer based on type).

Category Node Type – “category”

Category node type is container for html nodes (or children categories). It has the following properties:

- id: JCR_NODE_ID
- type: category
- title:** <Auto Generated based on node name>
- template: (not required, if missing default.properties have value pages/category.html)
- description:** <Admin Edited>
- order: <Admin Edited>

jcrCreated: (Creation Date)
jcrLastModified: (Modification Date)
metaKeywords: <Admin Edited>
metaDescription: <Admin Edited>

HTML Content Node Type – “html”

HTML content node type is used for rendering HTML content. Properties of this node type are:

id: JCR_NODE_ID
type: html
title: <Auto Generated based on node name>
template: (not required, if missing default.properties have value pages/view.html)
introText: <Admin Edited>
htmlText: <Admin Edited>
order: <Admin Edited>
images: Special property to access child images
files: Special property to access all child files
jcrCreated: (Creation Date)
jcrLastModified: (Modification Date)
metaKeywords: <Admin Edited>
metaDescription: <Admin Edited>

OSGi plugin concept

Developing complex application with java can be difficult especially if there are many external libraries referenced in a project. Similar like “dll-hell” in java world “jar-hell” is called situation where many jars are required and different version of same library required in different pieces of code. That is very common situation and OSGi comes to solve that problem (along with many others). It has automatic class loading features, managed life-cycle – install, uninstall, start, stop on bundle, and perhaps the most important – highly modular standard. There are no direct dependencies from one jar (bundle) to other. Software architect defines what one exports and what external classes will be used. All the rest is left on the framework which guarantees that all classes will be loaded correctly before running a bundle.

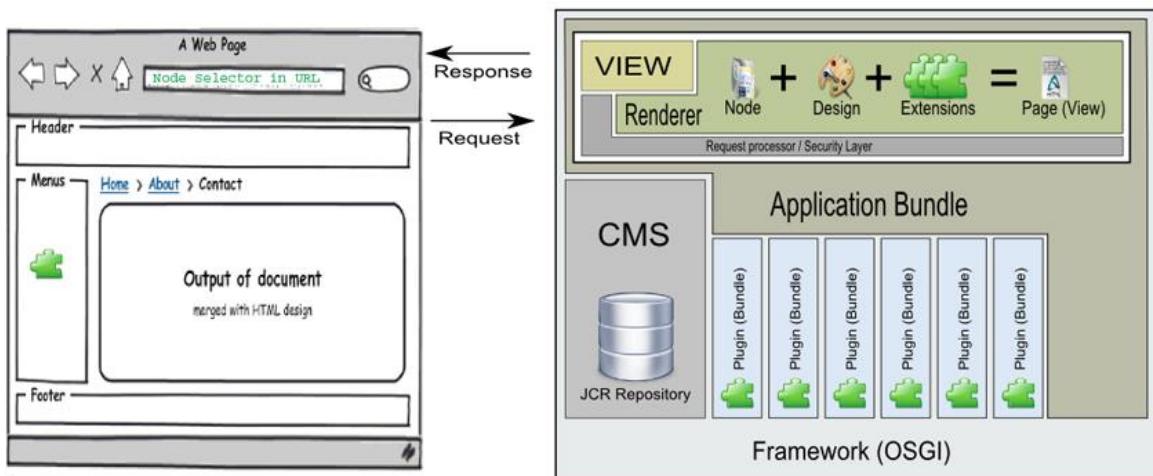


Figure 3: OSGi architecture in Chon CMS

CHON CMS DEVELOPMENT

Installation

Install Maven

Easiest way to build Chon CMS is with maven 3 (how to download and install maven 3 see <http://maven.apache.org/download.html>).

To check if maven is correctly installed type: mvn –version in your command line.

Generate Chon CMS Project Structure

To generate new Chon CMS based project, in command line type:

mvn com.choncms:chon-generate:new-project

```
C:\Documents and Settings\jovica.veljanovski>mvn com.choncms:chon-generate:new-project
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project <No POM> 1
[INFO] -----
[INFO]
[INFO] --- chon-generate:1.2:new-project (default-cli) @ standalone-pom ---
Enter project groupId - your chon based project name <root package>.
Value for 'Project groupId <Project Name>' (default <leave blank>): 'com.choncms.example': com.interworks
Oct 28, 2011 10:56:10 AM com.choncms.maven.UTemplate init
[INFO] Init velocity engine. Properties: {velocitymacro.context.localscope=true, url.resource.loader.modificationCheckInterval=10000}
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 8.159s
[INFO] Finished at: Fri Oct 28 10:56:11 CEST 2011
[INFO] Final Memory: 4M/8M
[INFO]
C:\Documents and Settings\jovica.veljanovski>cd com.interworks-chon
C:\Documents and Settings\jovica.veljanovski\com.interworks-chon>dir
Volume in drive C has no label.
Volume Serial Number is 68B3-753A

Directory of C:\Documents and Settings\jovica.veljanovski\com.interworks-chon

10/28/2011  10:56 AM    <DIR>          .
10/28/2011  10:56 AM    <DIR>          ..
10/28/2011  10:56 AM    <DIR>          app
10/28/2011  10:56 AM    <DIR>          bundles
10/28/2011  10:56 AM    <DIR>          doc
```

With this command maven will download all necessary plugins for generating new project. It will ask you for a project name (insert package based name eg: com.interworks), and it will generate new folder in current directory, com.interworks-chon. It will contain the following structure:

app - container for deployment structure- **DEPLOYMENT DIR**

- <PROJECT>-web – web container
- <PROJECT>-work.dir – repository+configurations+binaries from plugins

bundles – source code from plugins, put all plugins here – **SOURCE DIR**

- <PROJECT> - initial project plugin
- <PROJECT>-product – deployment definition, tells build system what plugins should be copied on deployment.
- pom.xml
- target-definition – Eclipse PDE helper for setting up platform

doc – project documentation – **DOCUMENTATION DIR**

Build

Navigate console to bundles dir and run the command:

mvn package

This will build the project with all necessary plugins. To check if plugins are built check work dir (app/<PROJECT>-work.dir/plugins), you should see all initial osgi bundles as jars built in this directory.

```
[INFO] Reactor Summary:
[INFO]
[INFO] bundles ..... SUCCESS [0.000s]
[INFO] com.interworks ..... SUCCESS [6.045s]
[INFO] com.interworks Target Definition ..... SUCCESS [0.478s]
[INFO] com.interworks product ..... SUCCESS [9.482s]
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 2:43.042s
[INFO] Finished at: Fri Oct 28 11:14:37 CEST 2011
[INFO] Final Memory: 11M/44M
[INFO]
C:\Documents and Settings\jovica.veljanovski\com.interworks-chon\bundles>cd ..\app\com.interworks-work.dir\plugins
C:\Documents and Settings\jovica.veljanovski\com.interworks-chon\app\com.interworks-work.dir\plugins>dir
Volume in drive C has no label.
Volume Serial Number is 68B3-753A

Directory of C:\Documents and Settings\jovica.veljanovski\com.interworks-chon\app\com.interworks-work.dir\plugins

10/28/2011  11:14 AM    <DIR>          .
10/28/2011  11:14 AM    <DIR>          ..
5,896  com.interworks-1.0.0-SNAPSHOT.jar
89,582  com.springsource.javax.servlet-2.5.0.jar
72,642  org.apache.felix.fileinstall-3.1.10.jar
404,211  org.apache.felix.framework-3.2.2.jar
8,401  org.apache.felix.http-api-2.2.0.jar
59,282  org.apache.felix.http.base-2.2.0.jar
63,812  org.apache.felix.http.bridge-2.2.0.jar
75,163  org.apache.felix.http.whiteboard-2.2.0.jar
2,199,116  org.chon.cms.admin-1.0.0.201108222116.jar
21,505  org.chon.cms.admin.editor-1.0.0.201108222116.jar
22,488  org.chon.cms.admin.explorer-1.0.0.201108222116.jar
2,568,449  org.chon.cms.content-1.0.0.201108222116.jar
52,295  org.chon.cms.core-1.0.0.201108222116.jar
25,073  org.chon.cms.menu-1.0.0.201108222116.jar
14,164  org.chon.cms.model-1.0.0.201108222116.jar
118,122  org.chon.cms.ui.jquery-1.0.0.201108222116.jar
116,832  org.chon.cms.ui.lightbox-1.0.0.201108222116.jar
10,995  org.chon.core.common-1.0.1.201108222116.jar
84,705  org.chon.core.utils-1.1.0.201108222116.jar
14,629  org.chon.jcr.client-1.0.1.201108222116.jar
52,872  org.chon.web-1.0.2.201108222116.jar
28,258,041  org.choncms.bnd-libs-1.0.0-SNAPSHOT.jar
10/28/2011  11:14 AM      514,295  osgi.compendium-4.1.0.jar
23 File(s)      34,852,569 bytes
2 Dir(s)   32,284,553,216 bytes free

C:\Documents and Settings\jovica.veljanovski\com.interworks-chon\app\com.interworks-work.dir\plugins>
```

Run

Run on Jetty server

Simple way to run is under jetty container using maven plugin. Go to app/<PROJECT>-web and type

```
mvn jetty:run
```

This will run jetty web server on default 8080 port.

You should now be able to access <http://localhost:8080/<PROJECT>-web>

Ready to change the template

Here is list of pages:

- [Main Category](#)
 - [Index Page in Category](#)
 - [Another page](#)
- [Category Info](#)
 - [About](#)

Path: /www
[Login](#)

Note: Default initial repository will automatically be created in app/<PROJECT>-work.dir/repository and simple 2-3 pages content will be inserted. After click on login you will be able to login with default user: admin/admin



Run on any application server as war

Web application can be packaged as war, app/<PROJECT>-web is very simple web application, just a simple starter for OSGi container. In the application in WEB-INF folder there is system.properties file that tells the application where the plugins are and how to run them. Modify app.base property and make sure all other properties (like repository dir, config dir etc) points to valid locations on local file system. You can build the war using maven war command and deploy the artifact on any web server.

Template development

The template files are located in your project plugin (bundles/<PROJECT>/tpl and bundles/<PROJECT>/res. In res folder you can find static resources (css, images, javascript), and tpl folder is the one where your velocity templates are located. By default Chon CMS will have 3 types on nodes (home page, category and html), and corresponding velocity templates are located in tpl/pages/, home.html, category.html and view.html.

HTML rendering starts with tpl/pages/base.html (for all nodes), and inside body your specific content is rendered based on your node type (home.html, category.html or view.html).

In templates 3 variables are passed:

- \$this – current node
- \$ctx – application context
- \$ext – extension map

Creating new plugin

To generate new plugin, navigate to bundles directory and use:

```
mvn com.choncms:chon-generate:new-plugin
```

```
C:\Documents and Settings\jovica.veljanovski\com.interworks-chon\bundles>mvn com.choncms:chon-generate:new-plugin
[INFO] Scanning for projects...
[WARNING] Some problems were encountered while building the effective model for com.interworks:com.interworks-product:eclipse-application:1.0.0-SNAPSHOT
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-install-plugin is missing. @ line 43, column 13
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING] No explicit target runtime environment configuration. Build is platform dependent.
[WARNING] No explicit target runtime environment configuration. Build is platform dependent.
[INFO] Reactor Summary:
[INFO]   bundles
[INFO]     com.interworks Target Definition
[INFO]     com.interworks product
[INFO]
[INFO] Building bundles 1.0.0-SNAPSHOT
[INFO] -----
[INFO] --- chon-generate:1.2:new-plugin <default-cli> @ bundles ---
*** Don't forget to add newly created bundle in modules in parent project to enable automatic build. ***
Please Enter fully qualified package name, e.g. com.choncms.example.my.plugin.one
Using Fully Qualified Package Name: com.interworks.visitor.counter
Oct 28, 2011 1:49:09 PM com.choncms.maven.UtTemplate init
INFO: Init velocity engine. Properties: {velocimacro.context.localscope=true, url.resource.loader.modificationCheckInterval=120, velocimacro.library=, output
[INFO]
[INFO] Reactor Summary:
[INFO]   bundles ..... SUCCESS [1:03.103s]
[INFO]   com.interworks ..... SKIPPED
[INFO]   com.interworks Target Definition ..... SKIPPED
[INFO]   com.interworks product ..... SKIPPED
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 1:37.622s
[INFO] Finished at: Fri-Oct 28 13:49:09 CEST 2011
[INFO] Final Memory: 10M/63M
[INFO]
C:\Documents and Settings\jovica.veljanovski\com.interworks-chon\bundles>dir
Volume in drive C has no label
Volume Serial Number is 68B3-753A
Directory of C:\Documents and Settings\jovica.veljanovski\com.interworks-chon\bundles

10/28/2011  01:49    <DIR>          .
10/28/2011  01:49    <DIR>          ..
10/28/2011  11:14    <DIR>          com.interworks
10/28/2011  11:14    <DIR>          com.interworks-product
10/28/2011  01:49    <DIR>          com.interworks.visitor.counter
10/28/2011  10:56    <DIR>          target-definition
1 File(s)           1,518 bytes
6 Dir(s)        32,281,722,880 bytes free
C:\Documents and Settings\jovica.veljanovski\com.interworks-chon\bundles>
```

Let's say we want very simple plugin that updates a counter when we get a hit on the page. When above mvn command asks for a name we would call it com.interworks.visitor.counter; that will generate new OSGi based plugin in bundles directory.

Note the creation structure of a plugin. It is fully compliant OSGi bundle that can be right after creation imported in eclipse and ready for development.

Best way to develop a plugin is to use eclipse PDE features. To setup eclipse see next section.

For counter plugin, update count code initial trigger would be in base.html since in that file all other templates are included. Each html node request is rendered using base.html. So, in base.html an extension can be called that updates simple counter stored in JCR.

Anywhere in base.html add code: \$ext.counter

What will happen in rendering pipeline: The framework will see if counter extension is registered (see app.registerExtension), get extension object (the class that implements org.chn.cms.core.Extension) and retrieve front end template object (call getTplObject on extension).

For counter only call to getTplObject can be used to update back end storage counter. Here is complete example how Extension implementation would look like:

```
package com.interworks.visitor.counter;

import java.util.Map;

import org.chon.cms.core.Extension;
import org.chon.cms.model.ContentModel;
import org.chon.cms.model.content.IContentNode;
import org.chon.cms.model.content.PropertyType;
import org.chon.web.api.Request;
import org.chon.web.api.Response;
import org.chon.web.mpac.Action;

public class CounterExtension implements Extension {

    @Override
    public Map<String, Action> getAdminActions() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Map<String, Action> getAjaxActions() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Object getTplObject(Request req, Response arg1, IContentNode arg2) {
        try {
            Long count = updateCount(req);
            return "Counter Updated; Current Count: " + count;
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return "Oops exception occurred: " + e.getMessage();
        }
    }

    private Long updateCount(Request req) throws Exception {
        //get content model
        ContentModel cm = (ContentModel) req.attr(ContentModel.KEY);

        //get node under /usr/local/counter - create if not exists
        IContentNode n = cm.getAppsConfigNode("counter", true);
        //set initial count to 0
        if(!n.getNode().hasProperty("count")) {
            n.getNode().setProperty("count", 0);
        }

        //update count
        Long currentCount = (Long) n.getPropertyAs("count", PropertyType.LONG);
        Long nextCount = currentCount+1;
        n.getNode().setProperty("count", nextCount);
        //save changes
        cm.getSession().save();
        return nextCount;
    }
}
```

Don't forget to register the extension in the framework:

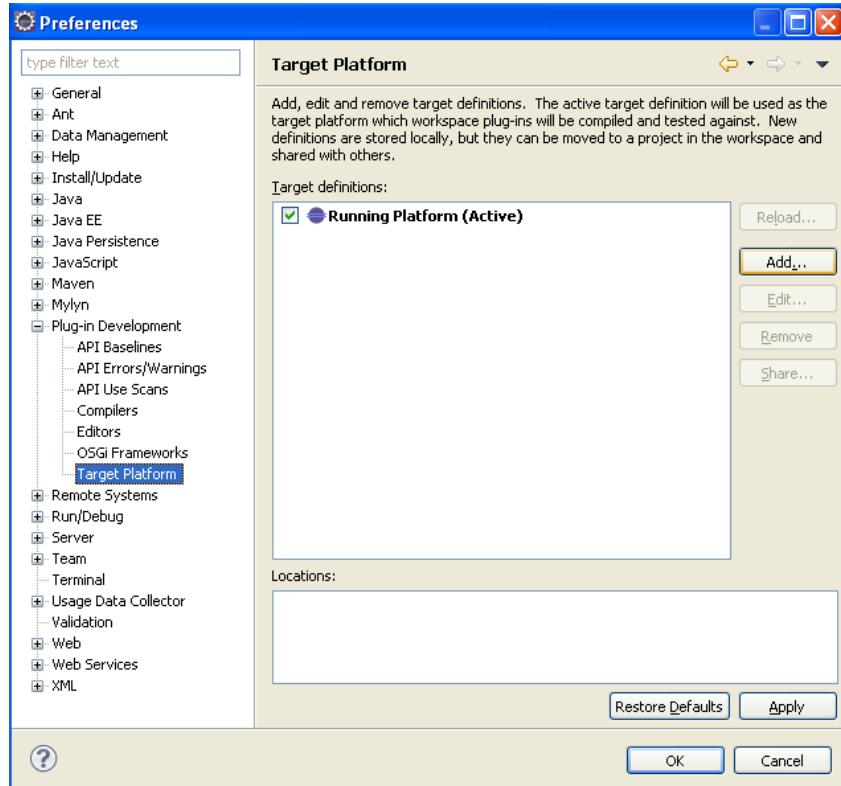
```
app.registerExtension("counter" new CounterExtension());
```

See next chapters for explanation what is "Content Model" and how to access to content nodes. That should explain code above.

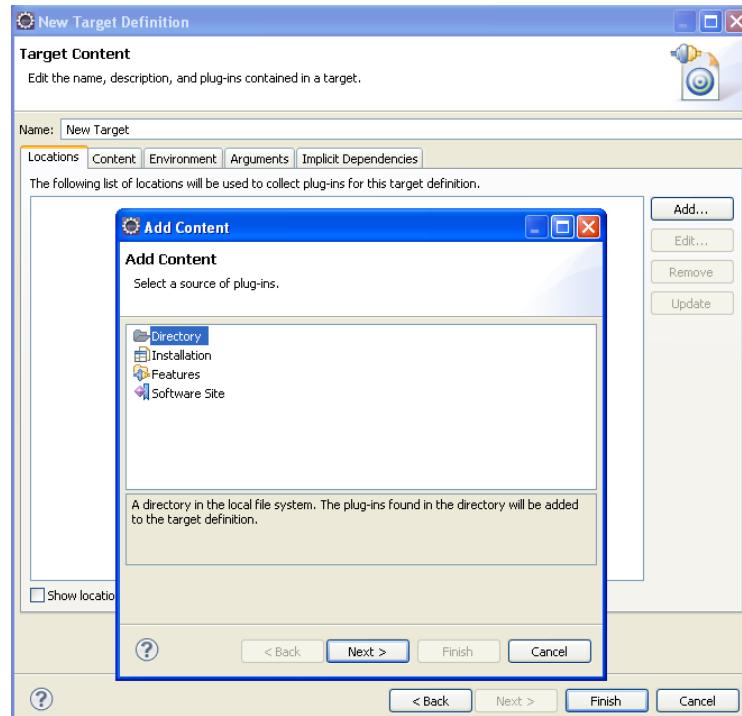
Setup Eclipse environment

Setting OSGi Target Platform

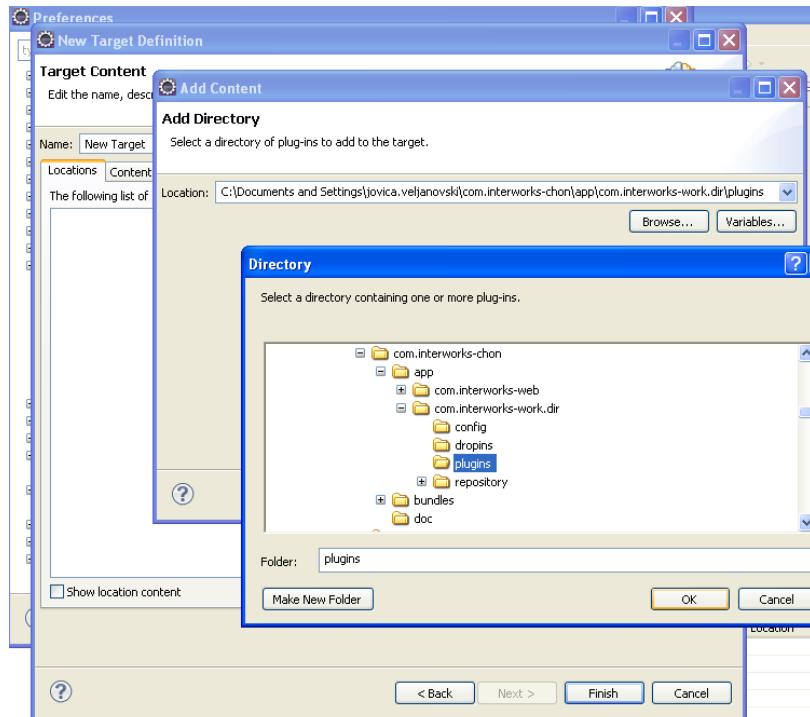
When developing OSGi based plugins eclipse needs to know on what “target platform” you are running the plugin. To set target platform using eclipse PDE go to (from eclipse menu) Window -> Preferences -> Plug-in Development -> Target Platform:



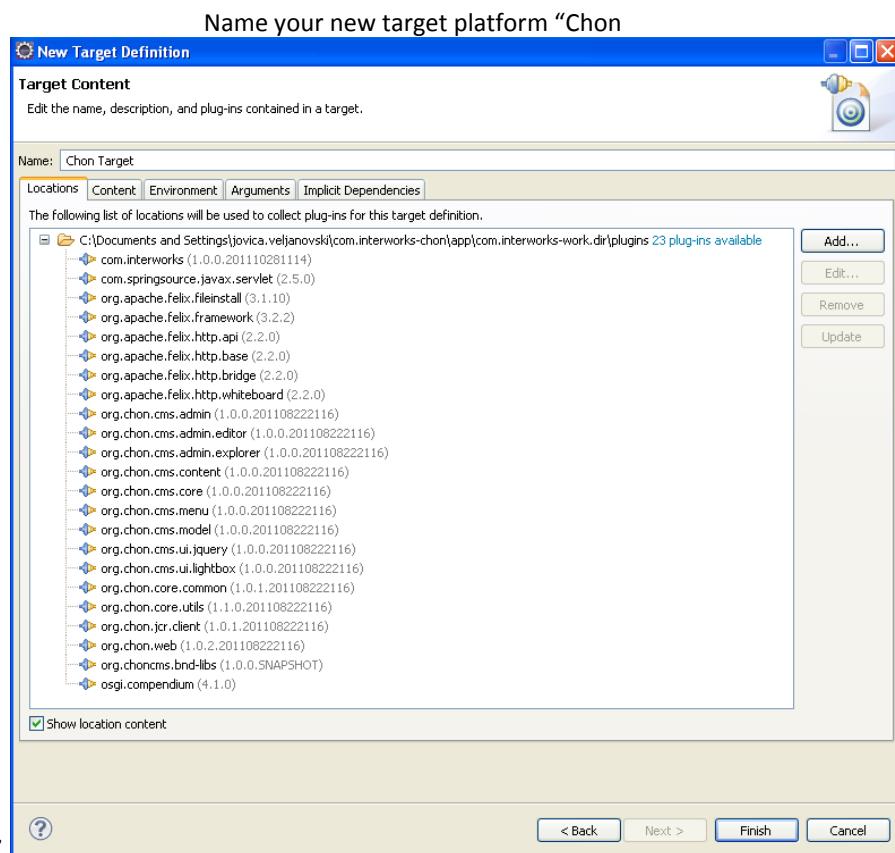
Click Add, select none template for platform creation (Nothing: Start from the beginning), and add your previously built plugins directory (app/<PROJECT>-work.dir/plugins).



This is similar to setting a class path for a new project. Eclipse needs to know all your other dependencies in order to build your plugin. Since in our case we need to develop Chon CMS based plugin it has dependencies from Chon CMS core, and that's why are need to tell eclipse pde about all our other plugins.

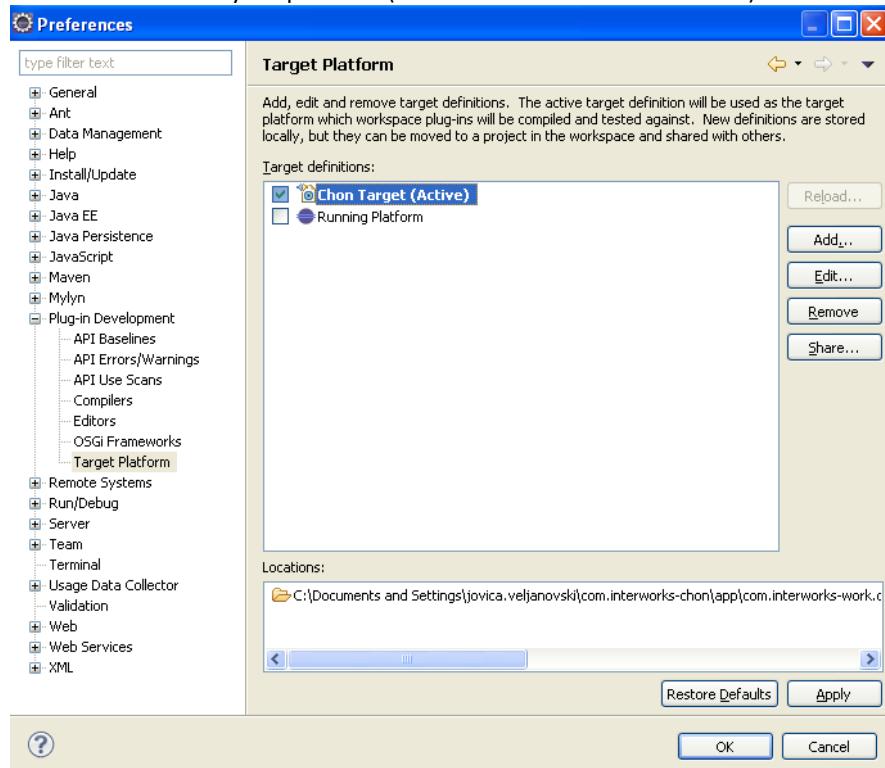


You should be able to select your plugins dir from your local file system. Eclipse PDE will find all OSGi plugins inside directory that you selected and it will count and display those (23 plugins available).



Click Finish.

Activate your platform (select check box after click finish).

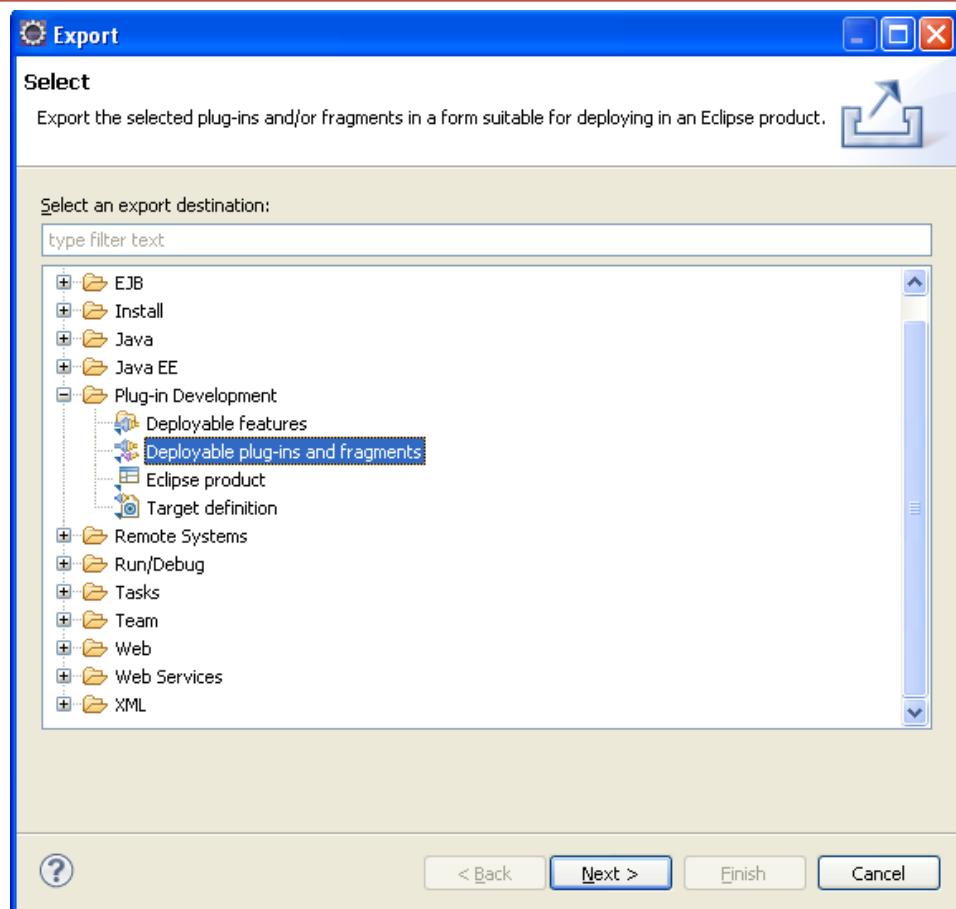


Importing Plug-in Projects

Plugins in bundles folder are already eclipse projects. After activation of “Chon Target” eclipse is able to recognize and set class path to all Chon CMS based plugins. At this stage using eclipse “Import Existing Projects” you are able to get development environment in eclipse.

Build Plugins from eclipse

Usually with maven setup plugins are built with maven. In development mode in eclipse you can build plugins using eclipse PDE export. Export will generate plugin jar and copy to specified location. To do this select plugin you want to export, from eclipse menu select File -> Export -> Deployable plug-ins and fragments:

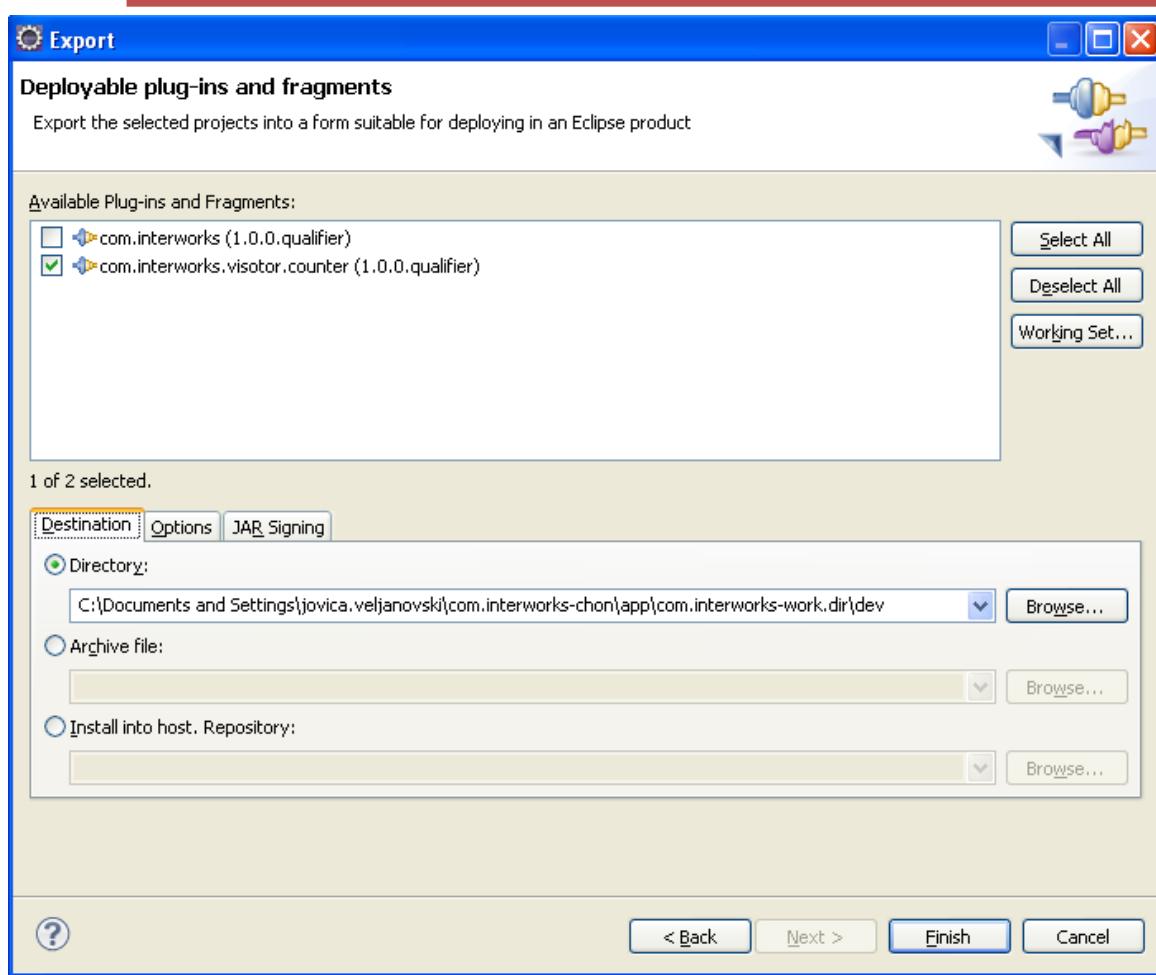


Next, select destination (Note, under destination folder, subfolder plugins is auto created by PDE). Select your work dir, dev subdir:

app/<PROJECT>-work.dir/dev

Above directory automatically is monitored for plugins changes. When you export plugins there they will be auto deployed and started. See system.properties configuration option:

```
#for development under eclipse, export plugins in ${app.work.dir}/dev  
chon.dropins.dir=${app.work.dir}/dev/plugins
```



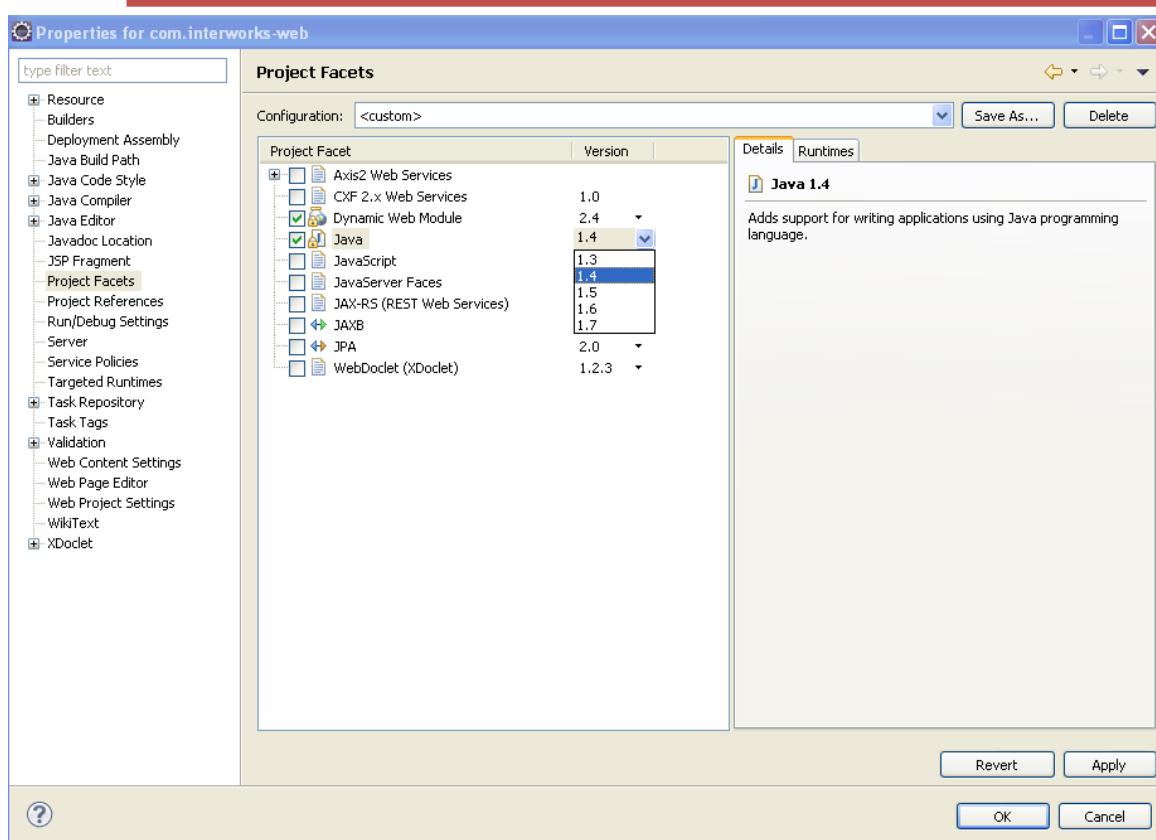
Important: Plugins are generated with default version selected from today's date. Select options tab and change "Qualifier Replacement" to SNAPSHOT – this way you ensure that your previous development build will be overridden by newest. If you don't change this, each export will generate new plugin and you will have duplicate bundles in the platform for each plugin.

Importing Web Application and configure tomcat web server

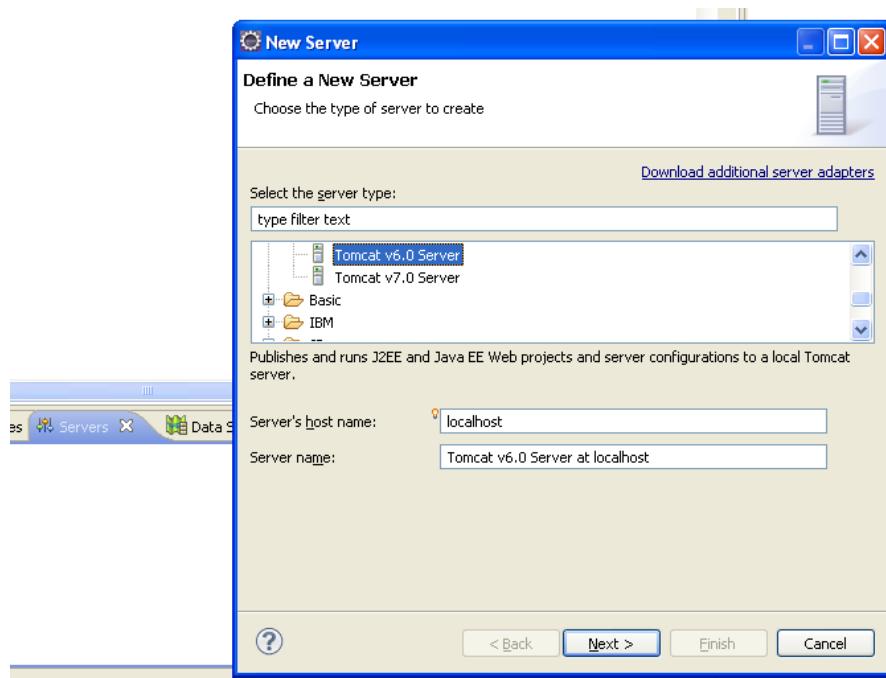
Although you can continue running the application with jetty, it is nice for debugging purposes to have web server run from eclipse. To do that use eclipse WTP plugin, generate eclipse project using maven:

In windows you can run app/<PROJECT>-web/mvn.eclipse.wtp.bat file. On other operating system take a look at command in mvn.eclipse.wtp.bat. It is standard mvn command using maven WTP plugin for generating eclipse project.

After creating eclipse project structure you can import as standard web application. Modify "Project Facets" to use java 1.5 since WTP plugin by default generates facet for java 1.4.



In servers you can now define tomcat (or whatever server you want to use for debug).



Modify system.properties (in WEB-INF dir), change app.base=.. to point to absolute path to your app:

```

Activator.java CounterExtension.java base.html system.properties

# Application name
app.name=com.interworks

# Base Application Directory, default: ${user.home}/chon-work-dir

# For Development Build, right above container,
# for prod deployment ${user.dir} will be used
app.base=C:/Documents and Settings/jovica.veljanovski/com.interworks-chon/app

app.work.dir=${app.base}/${app.name}-work.dir

# temp upload dir
uploadDir=${app.work.dir}/upload

# jackrabbit repository
repo.dir=${app.work.dir}/repository

# configuration dir and variables,
# autogenerated if not existing from __config files inside bundles
commons.configson.configuration.dir=${app.work.dir}/config

```

Run the server.

You should see notification in console for total plugins found and process of loading plugins:

```

Tomcat v6.0 Server at localhost [Apache Tomcat/6.0.13] C:\Program Files\Java\jre6\bin\javaw.exe (Oct 31, 2011 12:43:40 PM)
Oct 31, 2011 12:43:41 PM org.apache.catalina.core.AprLifecycleListener init
INFO: The Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: C:\Windows\bin\java
Oct 31, 2011 12:43:41 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-8080
Oct 31, 2011 12:43:41 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 293 ms
Oct 31, 2011 12:43:41 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Oct 31, 2011 12:43:41 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.13
Oct 31, 2011 12:43:41 PM org.apache.catalina.core.ApplicationContext log
INFO: ::>> Reading system.properties
Oct 31, 2011 12:43:41 PM org.apache.catalina.core.ApplicationContext log
SEVERE: Using felix-cache dir: C:\DOCUMENTS~1\JOVICA~1.VE\LOCALS~1\Temp\felix-cache
-----
----- Starting Chon Application -----
- Using work-dir ..... C:/Documents and Settings/jovica.veljanovski/com.interworks-chon/app/com.interworks-work.dir
- Plugins dir ..... C:/Documents and Settings/jovica.veljanovski/com.interworks-chon/app/com.interworks-work.dir/plugins
- Repository dir ..... C:/Documents and Settings/jovica.veljanovski/com.interworks-chon/app/com.interworks-work.dir/repository
- Site URL ..... http://localhost:8080/com.interworks-web
-----
Found 22 plugins. Loading ...
Oct 31, 2011 12:43:41 PM org.apache.catalina.core.ApplicationContext log
INFO: Installing bundle [file:/C:/Documents%20and%20Settings/jovica.veljanovski/com.interworks-chon/app/com.interworks-work.dir/plugins/com.spr
Oct 31, 2011 12:43:41 PM org.apache.catalina.core.ApplicationContext log
INFO: Installing bundle [file:/C:/Documents%20and%20Settings/jovica.veljanovski/com.interworks-chon/app/com.interworks-work.dir/plugins/org.apa

```

Note: Common error when running server is:

ERROR: Unable to start system bundle. (java.lang.Exception: Invalid plugins dir)

java.lang.Exception: Invalid plugins dir

That means that plugins dir is not configured well. Check system.properties, chon.plugins.dir=\${app.work.dir}/plugins, and make sure it points to a location where plugin packages are.

Plugin Development

Activator

In OSGi there is a concept of “bundle activator” – a class specified in MANIFEST.MF that has start and stop methods which are called when bundle is starting/stopping.

Plugins in Chon CMS should extend org.chon.cms.core.ResTpConfiguredActivator (when creating new plugin with mvn com.choncms:chon-generate:new-plugin this is default creation structure). In the activator there are 2 methods that should be implemented.

registerExtensions

getName

Similar like OSGi standard activator start method, registerExtensions is the method called when this plugin is activated. In this method you can initialize your model and create your extensions.

Example:

```
protected void registerExtensions(JCRAplication app) {  
    app.regExtension("counter", new CounterExtension());  
}
```

Extensions

Extensions are objects from class that implements interface org.chon.cms.core.Extension. In above explained activator a call to app.registerExtension, will automatically add new extension in the system identified with first argument – name and added in template variables in \$ext object. To access your extension “Template Object” (getTplObj) in velocity templates you will use extension name:

\$ext.counter

You have full access from templates to object created in your extension. You can use its methods, call its getters/setters, pass additional data etc.

Node Types

Each requested node is rendered based on type. Each type has NodeRenderer – class that defines how output of a node will be shown. By default all above types are rendered with Velocity Template Node Renderer (VTplNodeRenderer), the only difference is in their selection of a template. All 3 will use base.html, but each of those is rendered in its own template.

Repository setup file

Initially repository is created by definition from config/setup.kson. This file creates initial repository. If you don't have plugins for creating specific nodes you can “manually” create nodes using this file.

Configuration options

Besides system.properties – which contains properties for global configurations and running the framework, each plugin in Chon CMS can use json configuration file. Those files can be found in work.dir/config. Properties in those files can be accessed by getConfig method in ResTplConfiguredActivator. Each plugin has its own configuration properties, see plugin documentation for details.

Content Model

Content Model is interface that wraps JCR nodes and allows easy access to repository. There are 2 ways to access to content model:

In activator (when global application object is present):

- ContentModel cm = app.createContentModelInstance(getName());

In request (when code is in request context)

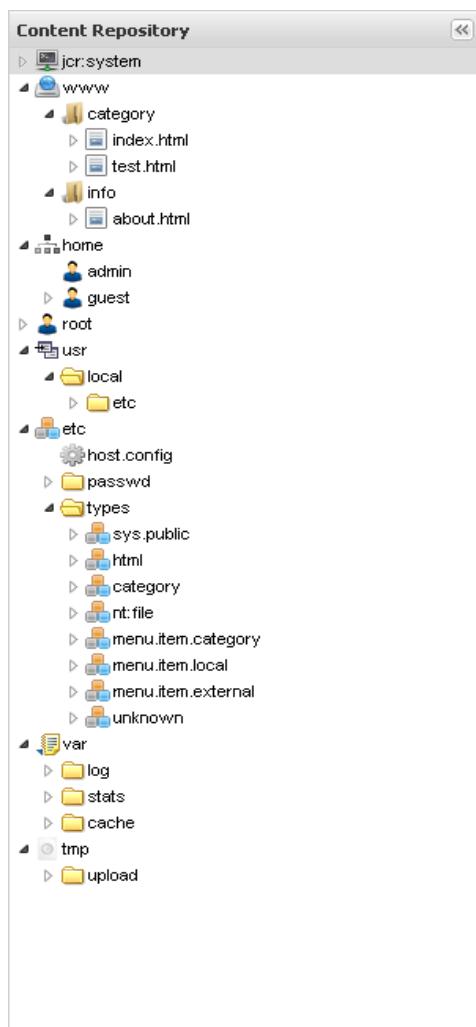
- ContentModel cm = (ContentModel) req.attr(ContentModel.KEY);

Content Model will wrap node based on its type and return class implementation for that node type. For example, cm.getPublicNode("/category/test.html") will return wrapped JCR node into HtmlContentNode which have additional properties like introText, htmlText (n.getIntroText(), n.getHtmlText()).

Generic interface for all content nodes is: IContentNode.

Initial repository structure

- New Rooting/dispatching is: chon application has all mappings from URL to Application.
- By default <http://domain.name/{app-name}> goes to workspace with name: {app-name}/www
- By default each workspace has a user with a name {app-name}-root
- And {app-name}-root has a default /root node in workspace
- App-names get only defined at bootstrap and regex of name can be [a-zA-Z0-9_-]
- By default: nobody and nothing can access nodes outside www except root user. And other users have in addition to www a /home/{user-name} dir.
- In /home/{user} the user can do whatever he wants to do
- An additional great service would be UserProfilePageService



Top level nodes are:

/www

Public Node, cm.getPublicNode(); For all direct public requests node is pick from this folder. For example, request to: http://<SITE_URL>/category/test.html will select /www/category/test.html node.

/home

Users Home Dir, cm.getUser() will give home node for currently logged in user.

/root

Root user home

/usr/{service-name}

All plugins should use this node to add plugin specific content. /usr/local/etc is special config dir where config nodes can be auto created. Use cm.getAppsConfigNode("menu") to get data from menu node.

/etc/{service-name}/ (as dir)

System Configurations, cm.getConfigNode(). /etc/types are all node types in the system. See "Node Type Definition"

/var

Folder for often changing files, eg. Logs, cache etc.

/tmp

Container for temporary nodes, cm.getTmpNode()

Repository initially is created using setup.kson configuration file.

Content model has methods for easy access to this creation structure. For example cm.getAppsConfigNode("mynode") will return /usr/local/etc/mynode.

Retrieve content node

In many cases you won't need Chon CMS specific content handling, you may just use JCR API to get, update and delete nodes, but if you use nodes created from ContentModel they will be wrapped to class you specify for particular node type. For example, nodes from type html will be wrapped in HtmlContentNode, nodes from type category in CategoryContentNode and so on.

There many ways to get node from content model:

Direct access, by absolute path

- `cm.getContentNode("/absolute/path/to/node")`

By relative path

- `cm.getAppsConfigNode("node")`: result node is /usr/local/etc/node
- `parentContentNode.getChild("node")`: result node is /path/to/parentNode/node

If you have reference from JCR node:

- `cm.getNode(reference)`: result is /abs/path/to/reference

Using JCR SQL query (wrap result reference JCR nodes in `cm.getNode()`)

- `cm.query("SELECT * FROM [nt:unstructured]")`

EXISTING PLUGINS

To enable Chon CMS work it comes with default plugins for simple content creation and management. Plugins can be grouped in 4 categories although this separate distinction doesn't always reflect the purpose of plugin. For example, Core utils plugin have administration page and can be used in UI.

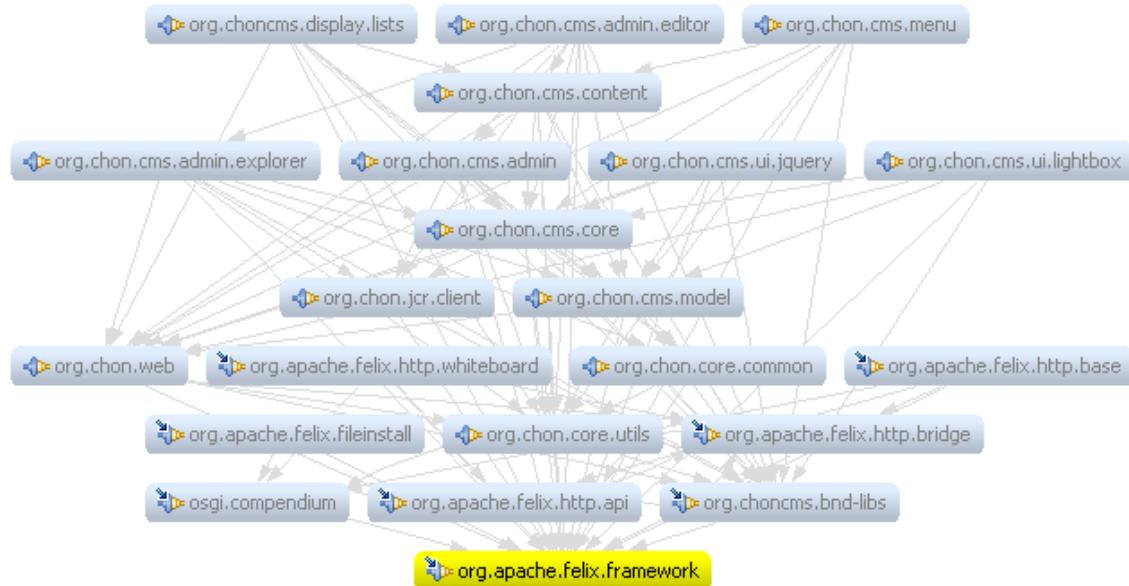


Figure 3: Visualization of all core plugins from Chon CMS

Visualizer for OSGi plugins (Figure 3) shows complex connections between all bundles in the system, nevertheless, when developing a plugin, build architecture adds default dependencies and programmer won't have to worry about backend complexity. For all dependencies framework will make sure correct connections are established as long as bundle metadata is correctly configured.

Third Party Plugins

Third party plugins used in Chon CMS are:

Felix Framework – [org.apache.felix.framework](#)

Felix framework is OSGi standard implementation. This bundle is container loader for all bundles - core OSGi bundle and provides plugins life cycle, context and base class loading.

OSGi services – [osgi.compendium](#)

Additional services used in OSGi.

Apache HTTP services – [org.apache.http.*](#)

Apache Felix HTTP services for java standard request handling.

Apache Felix Fileinstall – [org.apache.felix.fileinstall](#)

Fileinstall bundle is used for auto installing bundles in runtime. Usually this bundle is used in development to monitor a directory for new bundles or in runtime (in dropins) bundles can be dropped and auto started.

Core Plugins

Basic Chon CMS application can run only on core plugins. There will be no UI, only starting point for creating applications. Someone may find it useful to start from "core only" bundles to make application "from scratch" just as if building application on standard java servlets.

Core plugins are:

Legacy jars packed in OSGi bundle – org.choncms.bnd-libs

There are lots of dependencies that are used in Chon CMS that still are not ported to OSGi. All those are packed inside bnd-libs plugin. Here can be found common libraries, like apache.commons.logging, apache.fileutils etc...

Utils - org.chon.core.utils

Misc utility classes, for example access to database, org.json implementation etc.

Commons - org.chon.core.common

The purpose of this bundle is to load configuration files. It contains implementation for reading standard json configurations and all Chon CMS bundles can have one or many json config files auto loaded in their activators.

Web - org.chon.web

This bundle contains low level servlet processing. It has listener for servlet requests on all paths “/*” in application deployment context and process request to registered OSGi services as “Application”.

In this bundle are located all core classes for running the framework (org.chon.web.api):

- Application
- Request
- Response
- Resource
- ServerInfo

“Application” is the interface that is monitored in org.chon.web bundle. All services registered in OSGi context that implements “Application” are notified for requests. If “Application” returns a resource, that is sent for further processing.

In Chon CMS, there is only one “Application” service that handles all requests – “JCRApplication”. If necessary developer can have full control of request by registering another type of application and even remove Chon CMS plugins from request processing.

Request, Response and ServerInfo classes here are wrappers to standard servlet request/response classes. They contain methods for easy get parameters, easy write on response, nevertheless generic servlet request/response still can be used (req.getServletRequest(), resp.getServletResponse()).

Resource is interface that has only one method – process. Applications will return resource instance for given request (or null if resource not found). Resource implementation will take care of processing and returning the response. Resource can be seen as java HTTP servlet with difference that resource has only one function for all http methods (GET, POST).

Core - org.chon.cms.core

Core bundle is responsible to creating instance of Chon CMS “Application”. Instance of this application is called JCRApplication – the only “Application” service registered used for processing all requests and returns resources to web servlet.

Core bundle defines interface for Extension – simple UI extension points, and main JCR node wrappers.

Core bundle reads setup.kson file for initial setup of repository.

Model - org.chon.cms.model

Contains definition for content model – org.chon.cms.model.ContentModel

JCR Client - org.chon.jcr.client

This bundle is connector to JCR repository. It contains repository startup and repository OSGi service registration.

JCR Client bundle also contains utility class RepoService for easy access (CRUD) operations to repository (org.chon.jcr.client.service.RepoService).

Administration Plugins

Admin - org.chon.cms.admin

Admin plugin contains implementation for resource(s) under `http://<SITE_URL>/admin/*`.

Editor - org.chon.cms.admin.editor

Contains interface and registers OSGi services instances for various admin editors (html editor, category editor etc.)

Explorer - org.chon.cms.admin.explorer

Admin explorer component (default dashboard on admin part)

UI Plugins

Content Utils - org.chon.cms.content

Various utilities extensions:

Label Extension (\$ext.label) – simple key value map that can be modified by administrator.

Utils Extension (\$ext.utils) – extension for utilities in velocity

- formatDate(Calendar c, String pattern)
- escapeHTML(String html)

Display Lists - org.choncms.display.lists

This plugin allows administrators to create arbitrary collections of nodes. Each collection is identified by name and items can be used in template.

jQuery - org.chon.cms.ui.jquery

Lightbox - org.chon.cms.ui.lightbox

TECHNOLOGIES USED IN CHON CMS

OSGi

The Open Services Gateway initiative framework is a module system and service platform for the Java programming language that implements a complete and dynamic component model. Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot; management of Java packages/classes is specified in great detail. Application life cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly.

<http://en.wikipedia.org/wiki/OSGi>

<http://www.osgi.org>

<http://www.osgi.org/Specifications/HomePage>

Java Content Repository (JCR)

Content Repository API for Java (JCR) is a specification for a Java platform application programming interface (API) to access content repositories in a uniform manner. The content repositories are used in content management systems to keep the content data and also the metadata used in content management systems (CMS) such as versioning metadata. The specification was developed under the Java Community Process as JSR-170 (version 1) and as JSR-283 (version 2). The main Java package is javax.jcr.

A JCR is a type of object database tailored to storing, searching, and retrieving hierarchical data. The JCR API grew out of the needs of content management systems, which require storing documents and other binary objects with associated metadata; however, the API is applicable to many additional types of application. In addition to object storage, the JCR provides: APIs for versioning of data; transactions; observation of changes in data; and import or export of data to XML in a standard way.

http://en.wikipedia.org/wiki/Content_repository_API_for_Java

<http://www.day.com/specs/jcr/2.0/index.html>

<http://jackrabbit.apache.org>

Apache Felix

Apache Felix is a community effort to implement the "OSGi R4 Service Platform" and other interesting OSGi-related technologies under the Apache license. The OSGi specifications originally targeted embedded devices and home services gateways, but they are ideally suited for any project interested in the principles of modularity, component-orientation, and/or service-orientation. OSGi technology combines aspects of these aforementioned principles to define a dynamic service deployment framework that is amenable to remote management.

<http://felix.apache.org>

Apache Velocity Template

Velocity is a Java-based template engine. It permits anyone to use a simple yet powerful template language to reference objects defined in Java code.

When Velocity is used for web development, Web designers can work in parallel with Java programmers to develop web sites according to the Model-View-Controller (MVC) model, meaning that web page designers can focus solely on creating a site that looks good, and programmers can focus solely on writing top-notch code. Velocity separates Java code from the web pages, making the web site more maintainable over its lifespan and providing a viable alternative to Java Server Pages (JSPs) or PHP.

Velocity's capabilities reach well beyond the realm of the web; for example, it can be used to generate SQL, PostScript and XML (see Anakia for more information on XML transformations) from templates. It can be used either as a standalone utility for generating source code and reports, or as an integrated component of other systems. For instance, Velocity provides template services for the Turbine web application framework, together resulting in a view engine facilitating development of web applications according to a true MVC model.

<http://velocity.apache.org>

Eclipse PDE

The Plug-in Development Environment (PDE) provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and RCP products.

PDE also provides comprehensive OSGi tooling, which makes it an ideal environment for component programming, not just Eclipse plug-in development.

<http://www.eclipse.org/pde>

Tycho Maven Build

The build system for Chon CMS:

Tycho is focused on a Maven-centric, manifest-first approach to building Eclipse plug-ins, features, update sites, RCP applications and OSGi bundles. Tycho is a set of Maven plugins and extensions for building Eclipse plugins and OSGi bundles with Maven. Eclipse plugins and OSGi bundles have their own metadata for expressing dependencies, source folder locations, etc. that are normally found in a Maven POM. Tycho uses native metadata for Eclipse plugins and OSGi bundles and uses the POM to configure and drive the build. Tycho supports bundles, fragments, features, update site projects and RCP applications. Tycho also knows how to run JUnit test plugins using OSGi runtime and there is also support for sharing build results using Maven artifact repositories. Tycho plugins introduce new packaging types and the corresponding lifecycle bindings that allow Maven to use OSGi and Eclipse metadata during a Maven build. OSGi rules are used to resolve project dependencies and package visibility restrictions are honoured by the OSGi-aware JDT-based compiler plugin. Tycho will use OSGi metadata and OSGi rules to calculate project dependencies dynamically and injects them into the Maven project model at build time. Tycho supports all attributes supported by the Eclipse OSGi resolver (Require-Bundle, Import-Package, Eclipse-GenericRequire, etc). Tycho will use proper classpath access rules during compilation. Tycho supports all project types supported by PDE and will use PDE/JDT project metadata where possible. One important design goal in Tycho is to make sure there is no duplication of metadata between POM and OSGi metadata.

<http://www.eclipse.org/tycho>

Glossary

Activator: Implementation of OSGi BundleActivator – life-cycle manager of a bundle that contains “start” and “stop” methods. In Chon CMS, plugin activator usually extends JCRBundleActivator that has “registerExtensions” method called on start on bundle.

Content: In media production and publishing, content is information and experiences that may provide value for an end-user/audience in specific contexts. Content may be delivered via any medium such as the internet, television, and audio CDs, as well as live events such as conferences and stage performances. The word is used to identify and quantify various formats and genres of information as manageable value-adding components of media.

Content Model: Interface for easy access repository based on common structure created at startup. Nodes returned from this instance are wrapped into classes (based on node type) that implements IContentNode.

Node: Fundamental unit in repository.

Node Renderer: Piece of code that models output of a node. In Chon CMS node renderer implements interface INodeRenderer and instance of implementation is registered in OSGi context as service. Node Renderers are called automatically based on the type of node requested.

Repository: Tree like structure used for back-end storage.

Resource: In Chon CMS platform resource is abstraction for returning common output for each request. Resource can be static (image, file), can be node (from repository), can be servlet like class that can take full control of output etc.

Micro B3-IS Web Platform Security

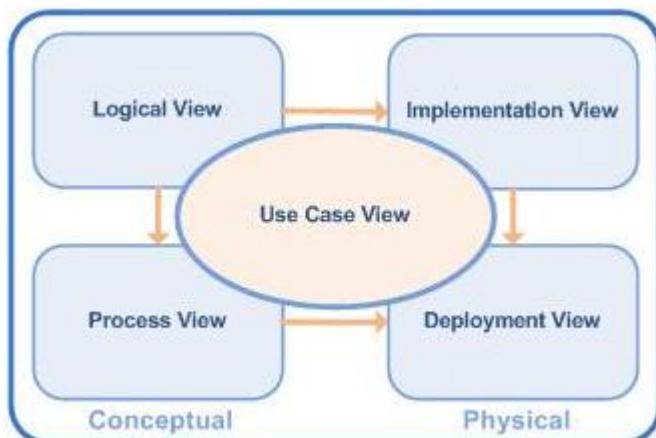
This document describes the security implementation of the Web Platform of Micro B3 Information System.

The security mechanism of the Micro B3 IS Web Platform checks every request made to the system and performs authentication and authorization. The mechanism itself consists of multiple sub-modules, each with distinct features and functionality:

- Data Model - this is the database model describing the users of Micro B3 IS and entities that support the security mechanism such as: roles, privileges, protected resources, consumers of resources, tokens.
- API for management of the security data model - this is a special sub-module that provides functionality for managing the security Data Model.
- Security Filter - filters and performs security check of each request sent to the Web Platform. Provides additional management user interface for subset of the entities in the Data Model (via the Security API)
- Applications Manager - provides administration user interface for managing the application that can use the Micro B3 IS Web Platform API.

Architecture

This section provides an architectural overview of the Security Module component. This Architecture Specification is created by applying the '4+1' view of architecture to the composite application. This approach organizes the architecture into views to meet the needs of individual stakeholders



This approach can be defined by documenting the following views of the architecture:

- **Use Case View:** describes the set of scenarios or use cases that represent some significant, central functionality of the system as seen by its end users or stakeholders in order to tie the other views together.
- **Logical View:** describes the application's functionality at a conceptual level in terms of its structural elements for analysis by architects, designers, and developers.

- **Process View:** describes an application in terms of the non-functional requirements (i.e. concurrency, scalability, and performance) as an extension to the Logical View and shows interactions between system components.
- **Implementation View:** describes the system at a physical level in terms of its components, layers, and subsystems for developers.
- **Deployment View:** describes the systems topology at a physical level by mapping the software onto the hardware and shows the distribution and communication during execution for deployment and release managers as well as infrastructure and support engineers.

Use Case View

This view describes the high-level use cases or scenarios of the solution in order to introduce the overall system architecture. This section will provide a functional context for the overall architecture by defining the actors and subsystems along with their relationships to the use cases.

Actors and subsystems

This section describes the high-level overview of the actors involved in the use cases of the Security Module.

The Security Module itself could be an actor in interaction with other modules of MB3-IS system, which is not in scope of this document.

Use Case Actors

MB3-IS user	User with permissions to create/update/delete client applications. In terms of oAuth – owner of the resources.
Client Application	Application with limited permissions. Allowed to use the portion of the Web API. In terms of oAuth – client acting on behalf of the owner.

Use Case Subsystems

1. Apps management module
2. Security Module Web API
3. Oauth provider endpoint module

Architecturally significant use cases

Architecturally significant use cases comprise a central function of the overall system or have broad architectural coverage. The purpose of these use cases is not to detail specification, but to provide bird-level view of the system from architectural stand point. The following architecturally significant use cases have been defined to provide context for the solution:

MB3-IS user authentication and Authorization Authentication of MB3-IS users in one of the authorization

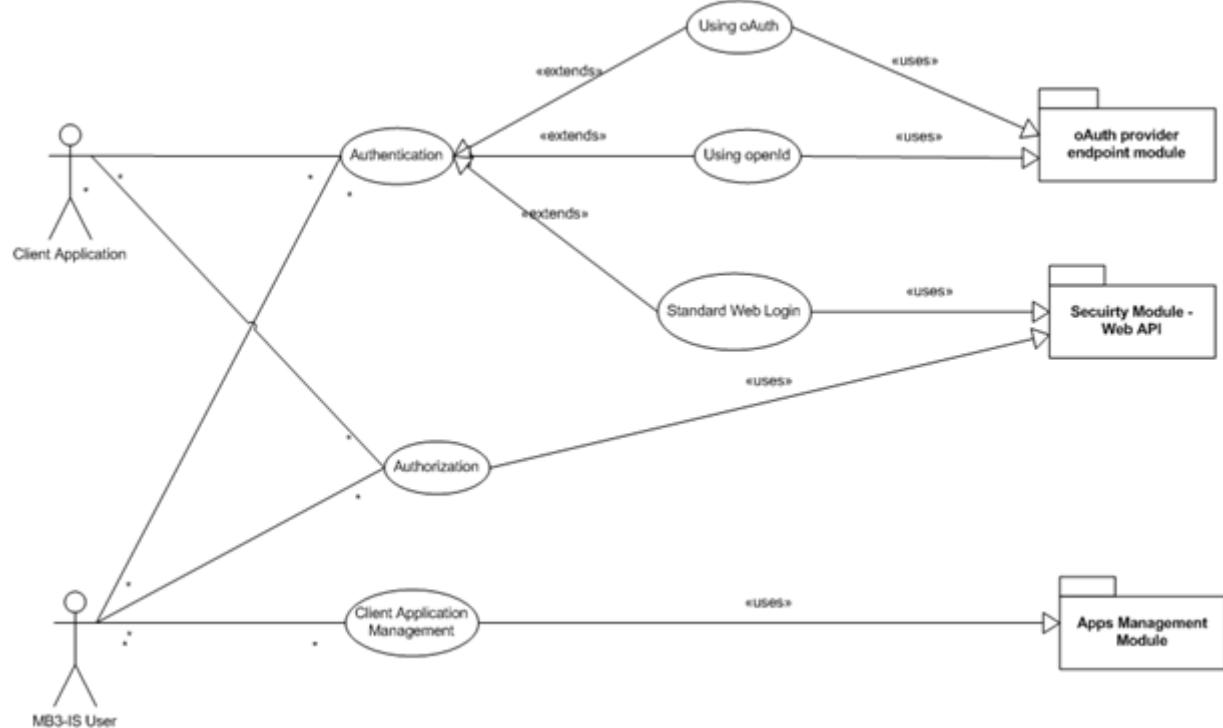
following ways:

- Standard web login
- Using external OAuth provider
- Using external opened provider

Client Application authentication Authentication of oauth consumers against MB3-IS and authorization oAuth provider mechanism

Client Application management CRUD management of consumer applications. Consumer Key-Secret management.

Use Case Relationships



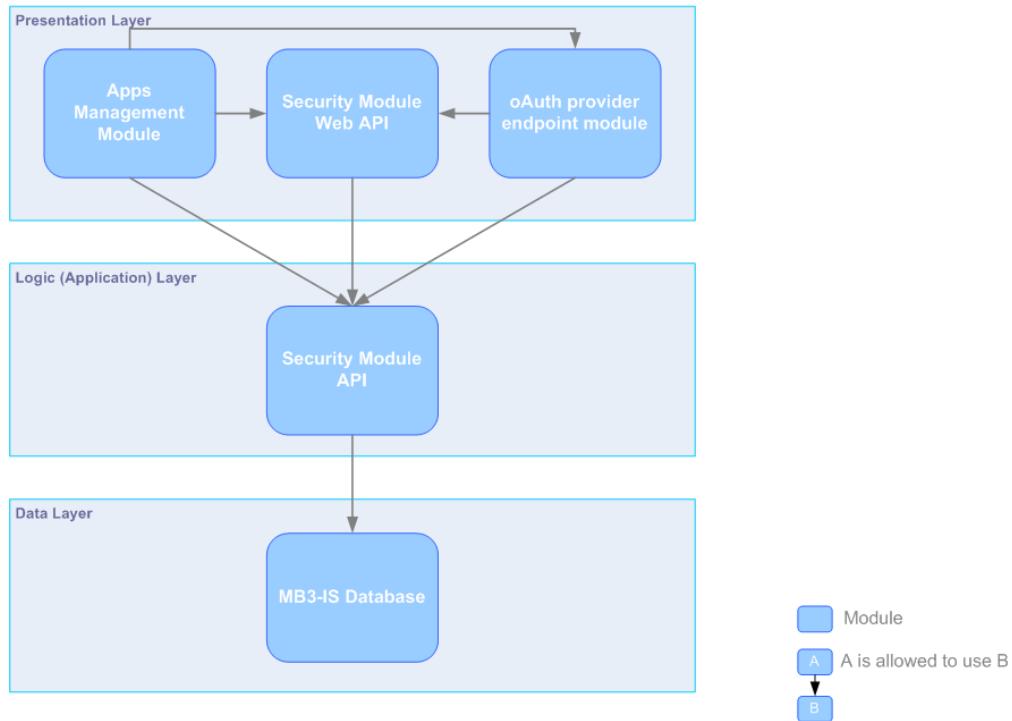
Logical View

The Security Module itself is a composite application defining modules in three layers:

- Presentation Layer: the modules in this layer comprise the interface for external interaction with the security aspect of the MB3-IS Web API. The Security Module offers the following packages in this layer:
 - Apps Management Module – management of the client applications (consumers) per user via web interface
 - Security Module Web API – authentication/authorization endpoints and services; web interface for standard web login, login using OAuth/openid via external providers and registration interface.
 - OAuth provider endpoint module – implementation of endpoints for OAuth provider as defined in the OAuth protocol
- Logic (Application) Layer: in this layer a single module is provided:

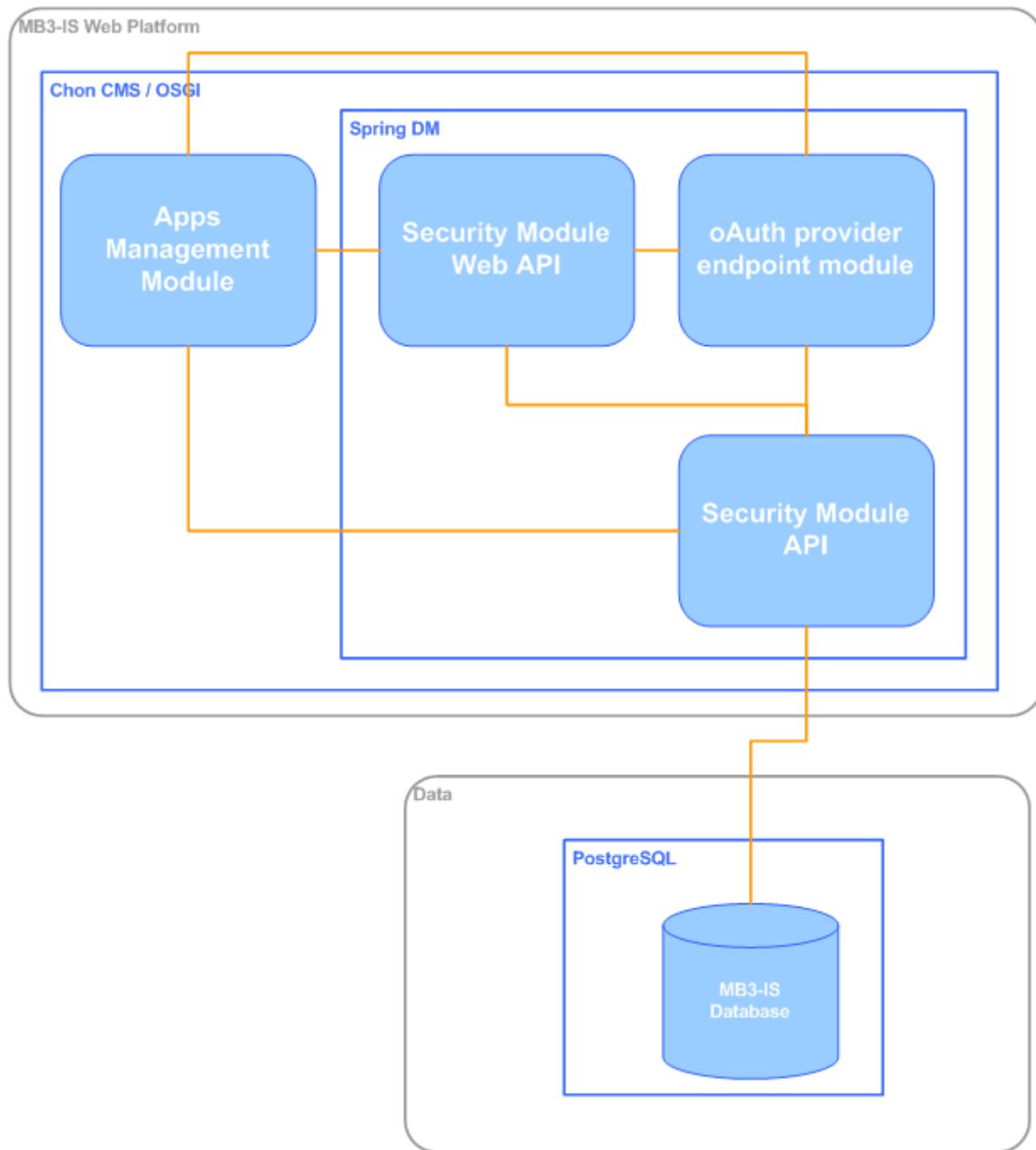
MB3-IS Security Management

- Security Module API – defines an API for management of users, authentication, authorization, oAuth specific functionalities such as:
 - Consumer applications management
 - Consumer key-secret management
 - Oauth tokens management
- Data Layer: this is the MB3-IS Database. Security specific information is stored in the MB3-IS database.



Logical Architecture

From logical view, the Security Module is a collection of sub-modules built on top of existing technologies (OSGI) and emerging technologies (Chon CMS, also built on top of OSGI). This architecture meets the requirements for modularity, extensibility, availability, maintainability.



In the current architecture of the Security Module four design packages are defined:

- Applications management module
- Security Module Web API
- oAuth Provider Endpoint Module, and
- Security Module API

Significant Design Packages

Applications Management Module

This module defines the means of managing applications that act as clients to MB3-IS Web API. The primary interaction between the application and the Web API is via

REST services. The authentication/authorization of the client applications is done using oAuth. This demands management of the client applications in the following aspects:

- Adding and removing client applications per user. A user of MB3-IS with sufficient privileges can register an application that will use the MB3-IS Web API. That application then can act on behalf of other users (as specified in oAuth) and can interact with MB3-IS Web API.
- Changing configuration for already registered application
- Management of consumer key-secret

The Applications Management Module is the web interface for the above management tasks.

Security Module Web API

This module defines the endpoints for authentication/authorization against MB3-IS, via standard login or using oAuth/opened in conjunction with external providers (multi-legged oAuth authentication/authorization). It also defines a web interface for user login and user registration.

oAuth Provider Endpoint Module

This module implements the oAuth web endpoints, as specified by the oAuth protocol for oAuth providers. Every time that an application would like to authenticate itself on Mb3-IS with oAuth protocol, it must use the endpoints provided by this module (such as generate request token, authenticate token, grant access page). This module uses the Security Module Web API for the following tasks:

- Login the user before the user can grant access to some consumer application

Security Module API

This module provides low-level programmatic API (service) to the security data model of the MB3-IS. In general, it provides the following services:

- User management services
- Consumers management services
- Authentication services
- Authorization services
- oAuth token management services

Process View

Operations

Entitlement

Entitlement in MB3-IS is a set of basic rules governing the access to the site, its resources such as application and forms, and its contents within forms based on user

MB3-IS Security Management

types. The concern of the Security Module as a subcomponent to the MB3-IS is primarily to authenticate and authorize the users and clients accessing MB3-IS.

The following are the defined user types for MB3-IS and consequently for the Security Module as well:

- Anonymous user
- Registered user – MB3-IS User
- MB3-IS Administrator
- Consumer Application – usually gets the privileges of the Owner of the resource which is Registered user on MB3-IS

The following processes are examples of when entitlement checks are performed:

- User Login - during user sign on, the Security Module checks the user credentials against the internal data store or against external OAuth providers, depending on the manner in which the user have chosen to login on MB3-IS.
- User Registration - During new user registration, entitlement rules will be checked to make sure that the user is allowed to be registered as a MB3-IS User.
- User Access to the Applications Management Module (Web Interface)
- Retrieving content from MB3-IS
- Interaction with the MB3-IS Web Services API – during interaction with the publicly exposed Web Services API of MB3-IS the entitlement rules are checked in accordance with OAuth protocol. Each of the publicly available services on the Web Services API require OAuth authentication/authorization.

Authentication

Authentication in the context of MB3-IS, is confirming the identity of a person or software program that require access to one or more resources on MB3-IS. This is one of the primary functions of the Security Module.

The authentication is performed in one of the following manners:

- Against local data-store and authentication source, – the user credentials are checked against stored credentials in MB3-IS Database.
- Against external authentication providers – the user authentication is obtained from external data sources using OAuth/openID protocol. Upon first use, some information is retrieved and stored in the local data-store. In case of this kind of authentication, the user credentials are not stored MB3-IS database.
- No authentication attempted – anonymous user. Every user that uses any of the web interfaces of MB3-IS and is not signed in the system, is considered to be anonymous user.

Authorization

Authorization determines which resources users have permissions to access. There are three levels of authorization pre-defined for MB3-IS:

- Anonymous access

MB3-IS Security Management

- MB3-IS User – every registered user has at least this level of authorization.
- MB3-IS Administrator – has administration rights.

Performance

The following are notes on the performance risks for the Security Module:

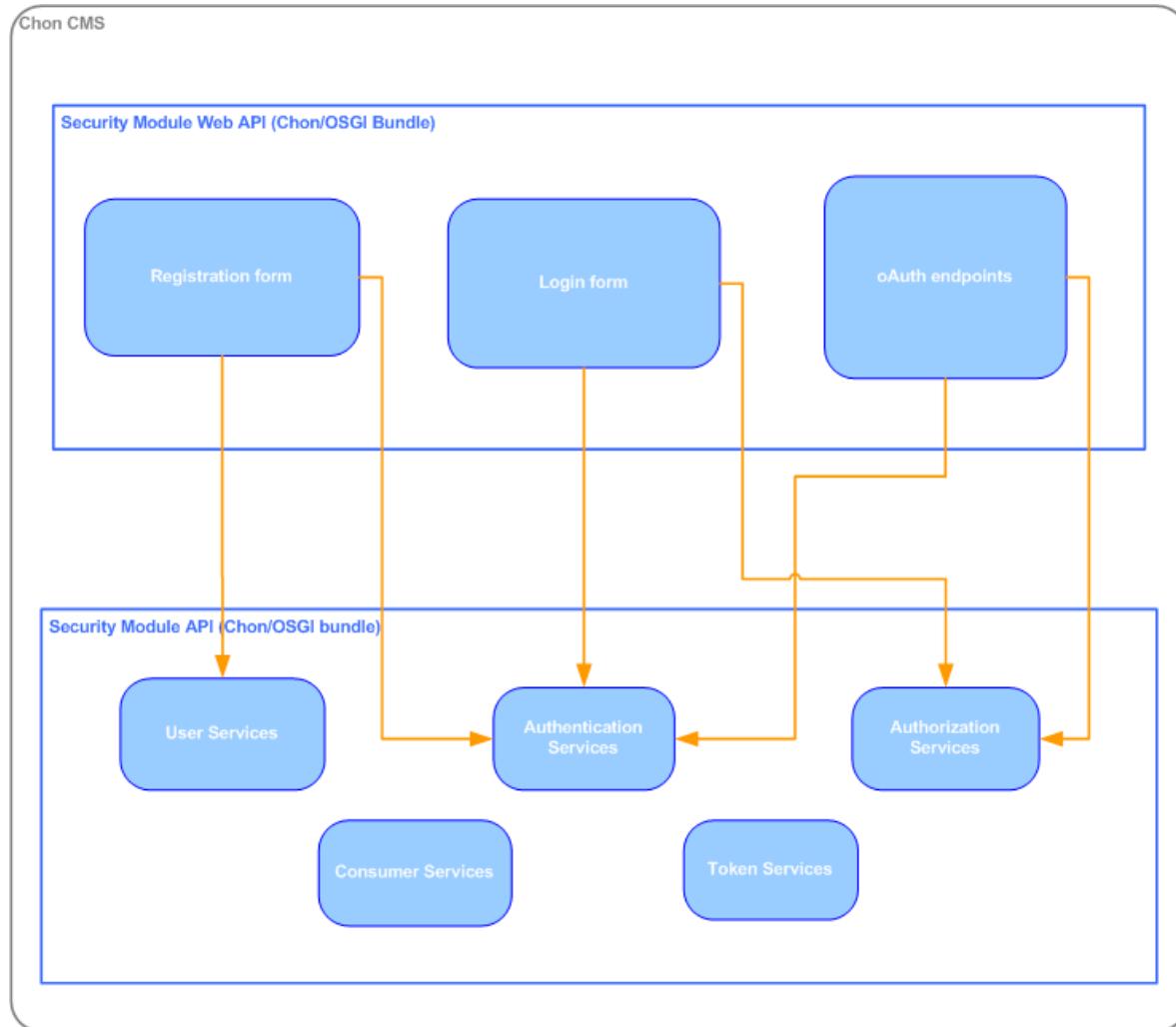
1. Token management – oAuth authentication/authorization protocol requires management of 2 types of tokens, request tokens and access tokens. The request tokens are short lived and exist only during the steps of the oAuth protocol. Once the client obtains access token, the request token is no longer used and required.

The access tokens have a longer period of validity and require persistence.

1. Web Service API usage – each call to the Web Services API need to go thru the Security Module:
 1. To authenticate the client if not already authenticated
 2. To confirm the validity of the signature for the request to the API resource

Implementation View

This section describes the physical system in terms of its subsystems, layers, and components.



Applications Management Module

Applications Management Module provides a user interface for managing client apps for MB3-Is Web Services API. It also provides an interface for consumer key-secret (client key) management per client app. The basic form of the Applications Management Module is a Chon Plugin.

Web Interface

The web interface of this module presents the user with the necessary forms for the following tasks:

- Preview on the client applications that he already registered
- View on the details for particular application
- Edit the information about the chosen application
- Add new app
- Remove a registered application
- Request new client key-secret pair
- Request access token for instant access

The web interface acts as the client part in a client-server architecture. The communication with the server – Chon CMS Plugin – is done via HTTPS – either standard web browser POST/GET or AJAX.

Applications management, Consumer key-secret management Chon Extensions

These extenstsions make up the backend for the Web Interface and provide basic CRUD functions for the registered applications and functions for retrieval of the consumer key-secret pairs and access token (re)generation.

Security

Because the data communicated between the web interface and the backed is sensitive – consumer key/secret pairs and access token key-secret pair, all of the communication has to use HTTPS.

Security Module Web API

The Security Module Web API provides standard endpoints for user login, user registration and standard endpoints for oAuth support.

From implementation point of view, it provides view templates (HTML) for:

- Login Page
 - Standard login
 - Login using external provider
- User Registration page

The login page templates are Chon templates – Velocity based templates with access to Chon specific context data. The registration page templates are Chon templates as well. The oAuth endpoints are HTTP endpoints implementing the callback interface for communication with external oauth providers (such as Google, Facebook, Twitter etc).

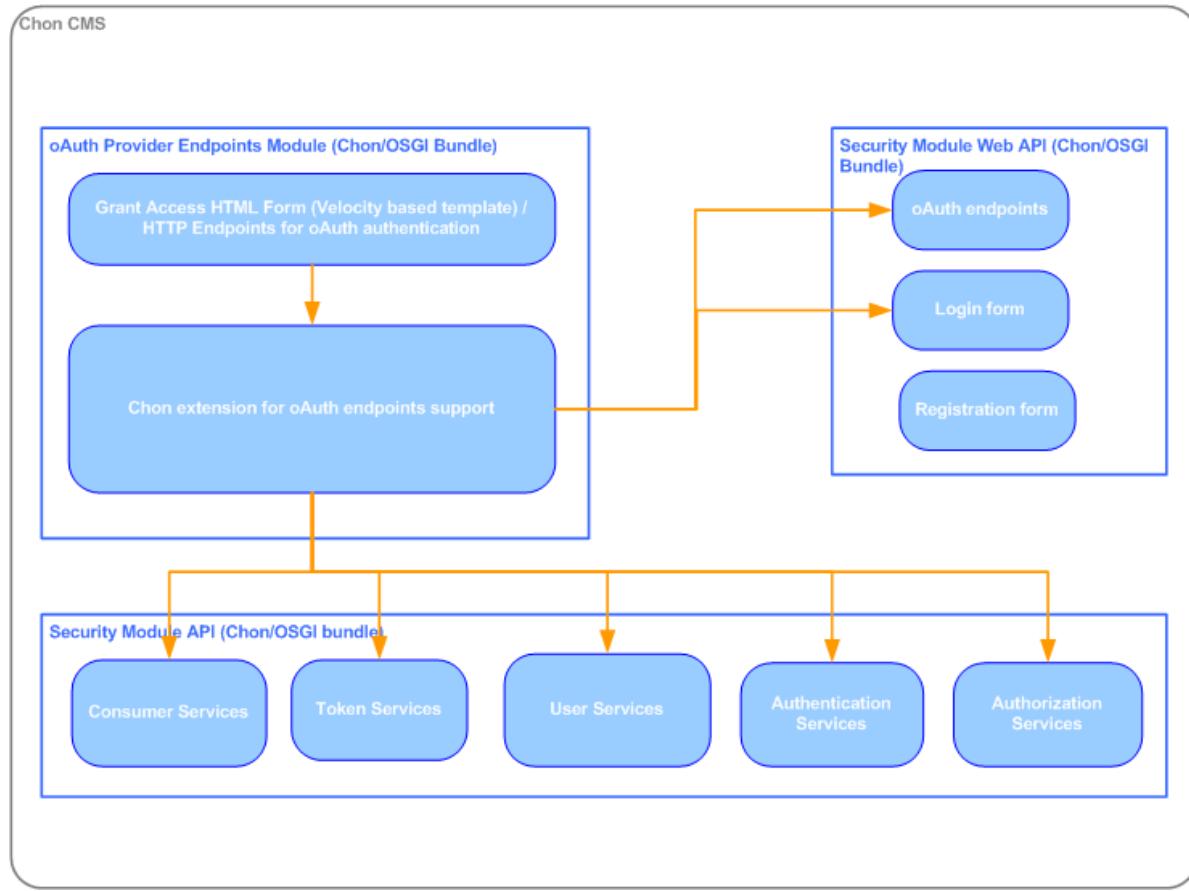
oAuth Provider Endpoint Module

This module provides support for oAuth Provider services. Implements the oAuth HTTP endpoints for:

- generating request tokens
- authorizing tokens
- template for granting permissions on client apps to specific resource on MB3-IS

The oAuth endpoints are HTTP endpoints implemented using Spring Security OAuth security filters.

The Grant Permissions page is a Chon template.



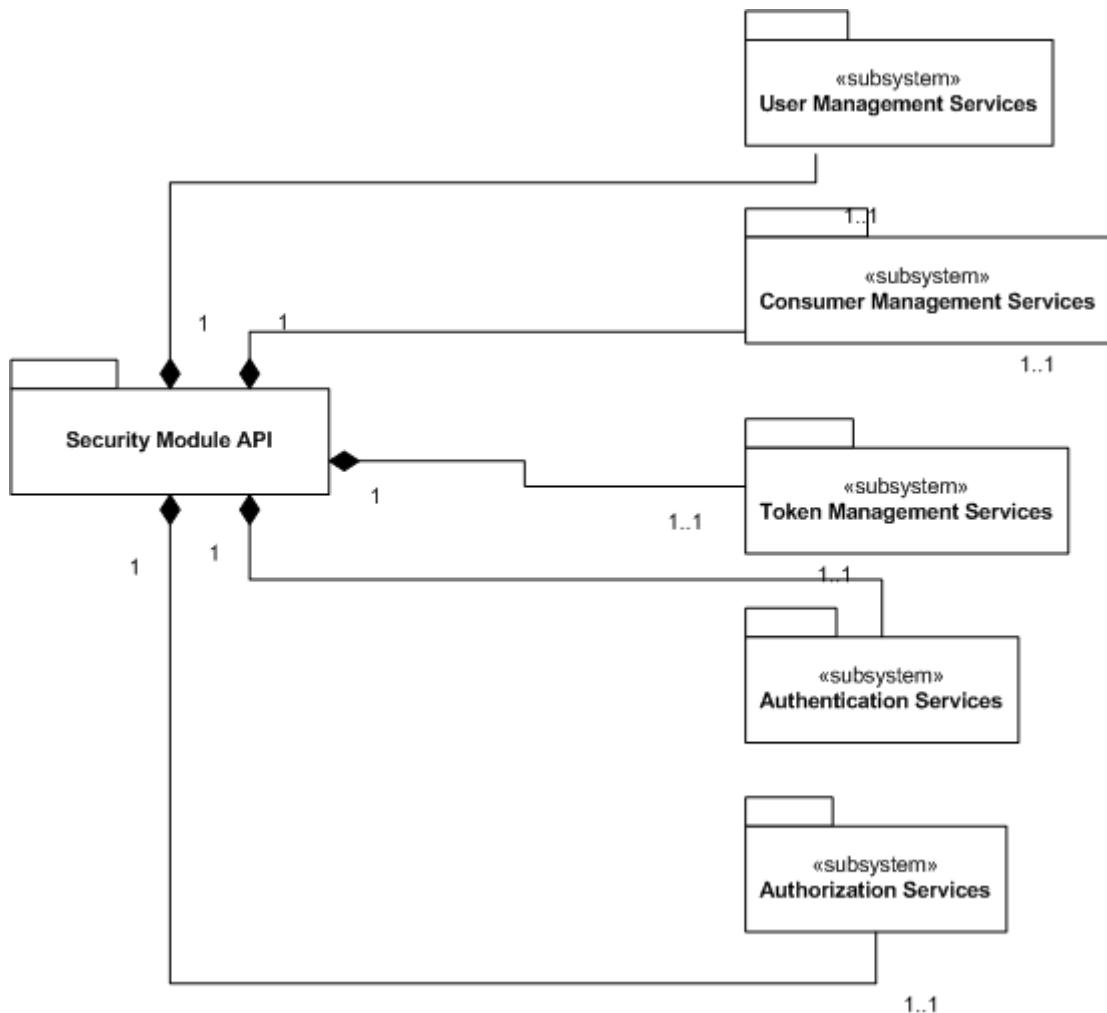
Security Module API

This module represents the low-level programmatic API that provides the essential interface to the security system of MB3-IS. The interface could be decomposed on the following logical modules:

- User Management Services – provide access to the MB3-IS users info, functionalities such as user info retrieval, adding/removing users or updating user information data.
- Consumer Management Services – provide access to the consumer information stored in MB3-IS – adding/removing consumers, retrieval of consumer information and updating the info about consumer application.
- Token Management Services – management of request/access tokens generated during the steps of the oAuth protocol.
- Authentication Services – provides authentication service. Although logically separated in this view, this service may be tightly coupled with the User Management Service during implementation.
- Authorization Services – provides services that determine the authorization level of a specified user. Although logically separated in this view, this service may be tightly coupled with the User Management Service during implementation.

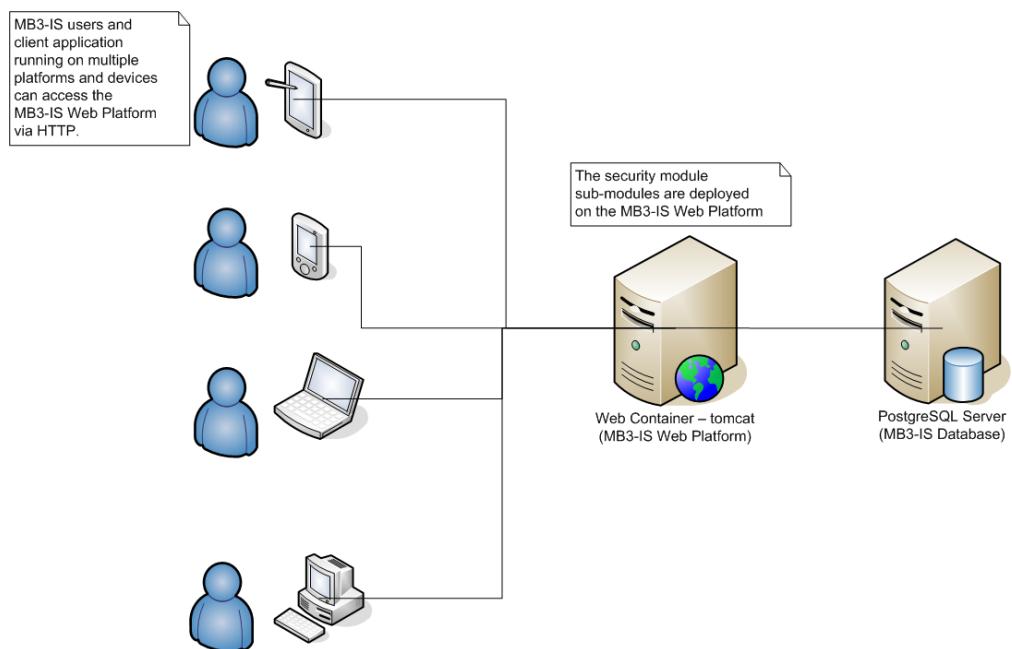
The following diagram shows the Security Module API Composition:

MB3-IS Security Management



Deployment View

This section describes the physical solution architecture in terms of the hardware and software.



Security Management Data Model

The Data Model consists of the following entities:

- User - the user as recognized by the Micro B3 IS. All of the users that have access to all or part of the system.
- Role - Specifies a role that a user can have in the system. One user can have multiple roles in the system.
- Permission - Each role have one or more permissions in the system. The permission defines what the user with some role can do within the system or over a specified resource.
- Web Resource Permission - defines a protected resource/resources accessible only by a specified role with some permission. The Web Resource can be defined with wildcard to enable aggregating a class of web resources on the Web Platform or a set of sub-resources. One Web Resource Permission defines only one type of access to the resource on the Web Platform - for example the resource "/protected/resource1" can be accessed read-only - i.e only using the "GET" HTTP verb, thus allowing for the resource to be retrieved, but cannot be changed (doing an HTTP POST, PUT or DELETE will result with access denied).
- Consumer - OAuth consumer entry. This specifies an application that some of the Users of Micro B3 IS has registered with the system to access some of the resources (web services).
- Access Token - is a OAuth Access Token that a Consumer can obtain to access some protected resource on the system in behalf of some User. The Access Tokens usually have long time to live (TTL) and are stored in the database until they expire.

The DDL script of the database (authdb) is as follows:

AuthDB DDL

```
drop schema if exists auth cascade;
create schema auth;

alter schema auth OWNER to megdb_admin;
set search_path = auth, pg_catalog;
set default_tablespace = '';
set default_with_oids = false;

-- functions
create or replace function auth.email_check(text) RETURNS boolean
language sql
AS
'select $1 ~* ''^[[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$'''';

-- tables
drop table if exists users;
create table users(
    logname text not null,
    first_name text,
    initials text,
    last_name text,
    description text,
    join_date timestamp with time zone not null,
    pass text,
```

MB3-IS Security Management

```
disabled boolean not null,
email text not null,
lastlogin timestamp with time zone,
constraint pk_users primary key (logname),
constraint valid_email_check check( auth.email_check(email) )
);

drop table if exists roles;
create table roles(
    label text not null,
    description text,
    constraint pk_roles primary key (label)
);

drop table if exists consumers;
create table consumers(
    key text not null,
    secret text not null,
    name text not null,
    description text,
    oob boolean not null default false,
    trusted boolean not null default false,
    expiration timestamp with time zone not null default now() + interval '1 year',
    logname text not null,
    callback_url text,
    constraint pk_consumers primary key (key),
    constraint fk_consumers_users foreign key (logname)
        references users(logname) on update cascade
        on delete cascade
);

drop table if exists access_tokens;
create table access_tokens(
    token text not null,
    secret text not null,
    verifier text,
    callback_url text,
    consumer_key text not null,
    user_log text not null,
    token_created timestamp with time zone,
    constraint pk_access_tokens primary key (token),
    constraint fk_access_tokens_consumers foreign key (consumer_key)
        references consumers(key) on update cascade
        on delete cascade,
    constraint fk_access_tokens_users foreign key (user_log)
        references users(logname) on update cascade
        on delete cascade
);

drop table if exists permissions;
create table permissions(
    label text not null,
    description text,
    constraint pk_permissions primary key (label)
);

drop table if exists web_resource_permissions;
create table web_resource_permissions(
    url_path text not null,
    http_method text not null,
    role text not null,
    constraint pk_web_resource_permissions
        primary key (url_path, http_method, role),
    constraint fk_web_resource_permissions_roles
        foreign key (role) references roles(label) on update cascade
        on delete cascade
);
```

```
-- ref permissions M---M roles
drop table if exists has_permissions;
create table has_permissions(
    role text not null,
    permission text not null,
    constraint pk_has_permissions primary key (role, permission),
    constraint fk_has_permissions_permissions
        foreign key (permission) references permissions(label)
            on update cascade
            on delete cascade,
    constraint fk_has_permissions_roles foreign key (role)
        references roles(label) on update cascade
            on delete cascade
);

-- ref users M---M roles
drop table if exists has_roles;
create table has_roles(
    role text not null,
    user_login text not null,
    constraint pk_has_roles primary key (role, user_login),
    constraint fk_has_roles_roles foreign key(role)
        references roles (label) on update cascade
            on delete cascade,
    constraint fk_has_roles_users foreign key (user_login)
        references users (logname) on update cascade
            on delete cascade
);
;
```

Applications Manager

Applications Manager is a web application, sub-module to Micro B3 IS Web Platform. It is implemented as Chon plugin – bundle: `net.megx.apps-manager`.

Once deployed, the apps manager is available under /apps on the Web Platform.

This application provides the following functionality:

- Add new consumer application for Micro B3 IS WEB API
- Update information about the registered application
- Remove registered application

Micro B3 IS WEB API and OAuth clients

Micro B3 IS Web API is a set of REST web services and resources. The API can be accessed only by authenticated and authorized applications.

For authentication and authorization process when accessing the API, OAuth (currently only version 1) is used. To access the API in this manner, the client (application) should conform to the following rules:

1. Must be OAuth capable - all the requests send to the API must be properly signed with the access token and secret

2. The application must be registered on Micro B3 IS Web Platform - in this way, the security mechanism can authenticate and authorize the application.
3. The application must have its own API Key and secret

OAuth1 endpoints

The server exposes 3 endpoints for OAuth1 authentication protocol:

- Request Token Endpoint - exposed on `/oauth/request_token`
- Authorization URL - exposed on `/oauth/authorize`
- Access Token Endpoint - exposed on `/oauth/access_token`

You must configure your client application with the above OAuth endpoints.

API Key

Once you've successfully registered your application, you will receive the API Key. You need to use the API Key to obtain access privileges to API. The API Key consists of two values:

- Consumer Key (or just key) - this is a value that is being sent to the server for obtaining the access token. Since this information is usually sent plain to the server, this is not a secret value. Also, because your application will have to use this value to obtain the access token more than once, you'll probably want to store this value in some configuration in the application itself.
- Consumer Secret - this is the shared secret between your application and the server. You'll use this value to sign requests and should never be send plain to the server. This is a secret value and needs to be kept safe in the application. Usually you need to securely persist both - the consumer key and the consumer secret.

The API Key for your application is stored on the server for as long as your application is used and registered on the server. If you choose to remove the application, the API Key will be removed and will not be restored back.

Access Tokens

The access tokens are issued to the application which wants to access some resource on the Web API. Your application can get Access Token only following the OAuth protocol.

Access Tokens are long lived, but do have an expiration date. If the access token expires, it cannot be used to access the API anymore and a new one must be obtained.

The Access Token consists of two values:

- Token - this is a string value and looks very similar in structure to the Consumer Key. It is sent plain with every request to the API. Because your

application will be sending it in subsequent calls to the API, you'll need to persist this value in-between calls.

- Secret - this is the shared secret between your application and the server. You need to keep this value safe.

The access token is issued per consumer and user, meaning:

1. The consumer client application can be used by more than one user
2. Each user can have different roles/permissions and different levels of access to the API

Therefore, different access tokens are issued for different users using your application. In this case, the Access Token is used to determine the user on which behalf your application acts.

Managing applications using the Web interface

To manage your applications, first navigate to the Applications Manager (/apps on the server).

You will be presented with the Applications Manager home page.

Apps Manager

The screenshot shows the 'Apps Manager' home page. On the left, there are two buttons: 'View Apps' (with the sub-instruction 'View your registered apps.') and 'Create an app' (with the sub-instruction 'Create new app'). On the right, there is a main content area with the following text:
Welcome to the Apps manager.
Please click on **View Apps** to view all of your apps, or click **Create App** to create your own App.

On the left you can see two buttons for the most common actions: "View Apps" and "Create an app".

If you click on "View Apps", you will be presented with the list of all applications that you have registered.

Apps Manager

The screenshot shows the 'Apps Manager' home page. On the left, there are two buttons: 'View Apps' (with the sub-instruction 'View your registered apps.') and 'Create an app' (with the sub-instruction 'Create new app'). On the right, there is a main content area showing a single application entry: 'MegxBar' (with the sub-instruction 'MegxBar Firefox extension') and a 'Details' link. There are also edit and delete icons in the top right corner of the card.

MB3-IS Security Management

To view the details of the application, click on the “Details” link. The information panel will be expanded in the current bubble.

Apps Manager

The screenshot shows the Apps Manager interface. On the left, there are two buttons: "View Apps" (View your registered apps.) and "Create an app" (Create new app). The main area displays the details for the "MegxBar" application. The application name is "MegxBar Firefox extension". Under the "Details" section, it shows "Desktop App (Out Of Band)": No and "Callback URL": none. A note states: "This is the application key and secret. Copy these and store them to safe place." Below this, the "App Key" and "Secret" values are listed. Further down, there is a section for "Access Token" with fields for "Token" and "Secret", and a "Generate" button. At the bottom right of the main panel are "Remove" and "Edit" buttons.

On this screen you can see a couple of values:

- App Key - this is the Consumer Key which is part of your API Key.
- Secret - this is the value for the Consumer Secret.

Both values need to be copied from here and stored to a safe place.

In the section below the API Key, you can request an Access Token for yourself. Click on "Generate" button and new Access Token will be generated for accessing the API. You can then copy the token values and insert them in your application (without going through the OAuth flow).

This is only available for the applications that you have registered. You cannot do this for applications that you have not registered.

This functionality is not intended as a general access mechanism, but instead may come in handy when developing/debugging new app and testing its access to the API without going thru the full OAuth flow.

Registering new applications

To register new applications, click on the "Create an app" button.

You will be presented with the form for application registration.

Apps Manager

View Apps View your registered apps.	Name: <input type="text" value="MegxBar"/> Description: <input type="text" value="MegxBar firefox extension"/> Desktop Client (Out Of Band): <input type="checkbox"/> Callback URL: <input type="text"/>	Cancel Save
Create an app Create new app		

The following fields need to be populated:

- Name - the name of the application. This is required field. The application name must be unique for Micro B3 IS.
- Description - optional. Description for your application.
- Desktop Client (Out Of Band) - check this option if your application is a desktop client application and cannot receive callbacks from the server.
- Callback URL - optional. This is the URL on which your application will be called after successful Authorization of the request token. This URL can also be specified as OAuth parameter in the first step of the OAuth protocol.

Click Save. Your application will be added to the list of applications.

Managing registered application

To edit the information about a previously registered application, find the application in the application list and click on "Details" to expand the details for the application. Click on the "Edit" button in the bottom of the panel.

Another option is to click the "Edit" button in the top right corner in the application entry.

You will be presented with form similar to the "Create new app" form.

Change the information and click "Save".

Removing an application

To remove application, find your application in the list of application.

Click on the "Close/Remove" button in the upper right corner of the application bubble, or expand the application details and click on "Remove" button.

Confirm the removal of the application.

The application will be removed from the system.

Using OAuth

OAuth basic flow

When using an HTTP client that needs to access oAuth protected resource, in order to access that resource the request itself must be signed using the Access Token issued by the resource holder (the server).

Every request made to the protected resource must be signed using this access token and a couple of other parameters as described in the oAuth version 1 protocol: <http://tools.ietf.org/html/rfc5849>.

The client can obtain the access token following the oauth protocol and basically doing the 3 steps described below:

1. Fetch Request Token - the client sends request signed with its Consumer Key/Secret to a special endpoint on the server (provider). The server validates the request, checks for existence of the consumer and issues a request token. The request token (token/secret pair) is returned to the client.
2. The client does a redirection to the Authorization URL. This is usually a GET request, signed with the request token and the consumer key and may contain a callback URL on which the server can then notify the client that the Request Token has been authorized and the client can now obtain the access token. The idea here is that the Resource Owner must manually authorize the client to access the protected resource. Note that the Resource Owner must have permissions (authorization) to access the protected resource. The Resource Owner must be authenticated against the server (usually by supplying credentials to the server) so the server can match that the Consumer (the client) acts on behalf of the authenticated Resource Owner. Following this chain of events, the user (Resource Owner) does not need to supply her credentials to the client, but only to the server that she trusts.
3. Obtain the access token. A request signed with the Request Token is sent to a special endpoint on the server. The server matches the Request Token and if it is authorized (step 2), an Access Token is issued to the client. The Access Token usually has longer time to live than a Request Token. The usual behavior of the client is to store the Access Token, which then can be used to sign every sub-sequential request to a protected resource on the server. The client needs to store this access token per user and it can reuse it in any time in the future. If the Access Token expires, the server will issue a security exception to the client when trying to access the resource, and the client has to go through steps 1 - 3 to obtain an Access Token again.

MB3-IS OAuth Endpoints

The OAuth endpoints provided by the security filter on Megx.net are as follows:

- Obtain request token URL: <megx.net_URL>/oauth/request_token
- Authorization URL: <megx.net_URL>/oauth/authorize
- Access token URL: <megx.net_URL>/oauth/access_token

So for example, in development mode, when megx.net is deployed locally, these URLs are as follows:

- Request Token: http://localhost:8080/megx.net/oauth/request_token
- Authorization: <http://localhost:8080/megx.net/oauth/authorize>
- Access Token: http://localhost:8080/megx.net/oauth/access_token

Example using Scribe

There is a test consumer in the repository implemented in Java using Scribe.

The consumer is implemented up to getting the access token.

Setting up the Consumer

The first thing is to set-up the endpoints, so the client will know on which URLs can obtain request token, where should it redirect for authorization and on which URL to request the access token.

This is the "API" class which will provide the builder with the endpoints:

MegxNetAPI.java

```
package net.megx.test.consumer;
import org.scribe.builder.api.DefaultApi10a;
import org.scribe.model.Token;
public class MegxNetAPI extends DefaultApi10a{
    private String requestTokenEndpoint = "http://localhost:8080/megx.net/oauth/request_token";
    private String accessTokenEndpoint = "http://localhost:8080/megx.net/oauth/access_token";
    private String authorizationURL = "http://localhost:8080/megx.net/oauth/authorize";

    @Override
    public String getRequestTokenEndpoint() {
        return requestTokenEndpoint;
    }
    @Override
    public String getAccessTokenEndpoint() {
        return accessTokenEndpoint;
    }
    @Override
    public String getAuthorizationUrl(Token requestToken) {
        return authorizationURL+"?oauth_token="+requestToken.getToken();
    }
    public MegxNetAPI(String requestTokenEndpoint, String accessTokenEndpoint,
                      String authorizationURL) {
        super();
        this.requestTokenEndpoint = requestTokenEndpoint;
        this.accessTokenEndpoint = accessTokenEndpoint;
        this.authorizationURL = authorizationURL;
    }
}
```

Then, we use this API to create the OAuth service:

OAuth Service initialization

```
OAuthService service = new ServiceBuilder()
    .provider(new MegxNetAPI(
        requestTokenEndpoint,
        accessTokenEndpoint,
```

```
        authorizationURL))
    .apiKey(apiKey)
    .apiSecret(apiSecret)
    .callback(request.getRequestURL().toString()+"?isCallback=true") // we're setting up the callback here - ommit this if the client cannot be called back - like for desktop clients
    .build();
```

Obtain Request Token

Obtaining request token is straight forward:

Obtain Request Token

```
Token requestToken = service.getRequestToken();
```

Authorize the consumer

The next step is to authorize the consumer. For this, we'll need to get the calculated authorization URL - which is basically the authorization endpoint URL + the request token as GET parameter called "oauth_token":

Authorize Request

```
String authURL = service.getAuthorizationUrl(requestToken);
// redirect the client to authURL here, or open it in browser
```

Obtain Access Token

Once the request token has been authorized, we can get the Access token as follows:

Get Access token:

```
//Verifier verifier = new Verifier("some-value");
Verifier verifier = null;
Token accessToken = service.getAccessToken(requestToken, verifier);
```

Note that the verifier is only used if the service responds with PIN, which the user has to enter manually in the client application. Since the Security filter authorizes the request token itself, the verifier is not necessary.

Get a protected resource

To get a protected resource, the access is the same as regular HTTP request. The only difference is that this request has to be signed with the access token obtained in the previous step.

With Scribe, this is as follows:

Get Protected Resource

```
String resourceURL = "http://localhost:8080/megx.net/pubmap/someResource"; // this is the URL of the REST resource we need to "GET"
OAuthRequest authRequest = new OAuthRequest(Verb.GET, resourceURL); // we're telling Scribe that we're using "GET" HTTP verb for this request

service.signRequest(accessToken, authRequest); // this is the actual sign of the request we're going to send
```

```
Response resourceResponse = authRequest.send(); // if everything went fine - we'll get a response, which has the InputStream to our protected resource.
```

JavaScript example using jsOauth

jsOauth is a JavaScript client library for oAuth version 1 and 2.

Links:

- On Github: <https://github.com/bytespider/jsOAuth>
- Documentation: <http://bytespider.github.com/jsOAuth/>
- API Reference: <http://bytespider.github.com/jsOAuth/api-reference/>

Creating a client (oAuth consumer service)

First you need to include the dependency to jsOauth:

```
<head>
    <script type="text/javascript" src="js/jsOAuth-1.3.4.js"></script>
....
</head>
```

In order to create an OAuth consumer (client) the following parameters must be provided:

- Consumer Key
- Consumer Secret
- Request Token URL
- Authorization URL
- Access Token URL

The instantiation of an OAuth service is done as follows:

```
var OAUTH_BASE = 'http://localhost:8080/megx.net-web/oauth'; // the base URL under which the oAuth endpoints can be found

var service = new OAuth({
    consumerKey: 'NzA1NDZhMTktZmMwOC00NmI2LTg0ZTUtNDg4ZWRmODE0ZjYx', // Consumer Key
    consumerSecret: 'NIaWgKMJOkAYm6vdFpgGHpPOFF4mXFmvqL-xcxb698hsid3y-bZ5FyGGMD7z0pqFk0w70l2qD7n-GdJ-HeW3Srqq', // Consumer Secret
    requestTokenUrl: OAUTH_BASE + '/request_token', // obtain request token endpoint
    authorizationUrl: OAUTH_BASE + '/authorize', // authorize the request token endpoint
    accessTokenUrl: OAUTH_BASE + '/access_token', // get access token from request token endpoint
});
```

Fetch Request Token

```
service.fetchRequestToken(function(url){
    // this is the success callback. this function gets called if the
    // request token has been successfully fetched.
```

MB3-IS Security Management

```
// The next step is to call the authorization URL, so the user can
authorize the Request Token
    window.open(url); // hitting the authorization endpoint.
}, function(errData) {
    // an error has occurred while fetching the request token
    alert("Error has occurred. The client cannot fetch request token");
});
```

Fetch Access Token

```
service.fetchAccessToken(function() {
    // the access token has been successfully retrieved.
    // Usually here the client needs to store the access token
}, function(errData) {
    // an error has occurred while fetching the access token
    alert("Error has occurred. The client cannot fetch access token");
});
```

Access a protected resource on the server

This step assumes that the client has the access token available.

If the Access Token is not available the client need to go through the above steps, again (fetch request token/authorize/fetch access token):

```
// If the access token was persisted, load it and feed it to the service
// NOTE: This step is unnecessary if the access token is fetched as described above
// (using the full protocol).
if(accessTokenAvailable()){
    var accessToken = loadPersistedAccessToken();
    service.setAccessToken(accessToken.token, accessToken.secret);
}

service.get('http://resource-keeper.org/resources/protected/secureResource',
function(response){
    // we have successfully accessed the protected resource.
    // the resource is available as a string value in the response, as such:
    alert('The protected resource: ' + response.text);
},function(err){
    alert('Failed to fetch the protected resource.')
});
```

Users Administration

The submodule Security Filter provides a Web interface for administration of the security for Micro B3 IS Web Platform. It offers user management and administration of the protected resources.

The administration intrfce is available on the following URL on the deployed Web Platform (megx.net): **http://<deployment domain>/<deployment context>/security/admin**, where:

- **deployment domain** is the domain on which Micro B3 IS Web Platform is deployed
- **deployment context** is the Application context of the Web Platform (may be omitted)

Users Administration provides basic management of the users. The following functionality is available:

- Add User
- Remove/Delete User
- Update User info
- View Users

Upon entering the Security Administration console, you will be presented with the list of users registered in the system.

The screenshot shows a user interface titled 'Manage Users'. On the left, there is a sidebar with two items: 'Users' (which is selected and highlighted in blue) and 'Web Resources'. The main content area is titled 'Manage Users' and contains a table with two rows. The first row has a cell containing 'Megx.net Dot Net' and another cell with an 'X' button. The second row has a cell containing 'Pavle Jonoski' and another cell with an 'X' button. In the top right corner of the main content area, there is a blue button labeled 'Add User' with a '+' sign.

From this list, you can choose to edit the info about some user by clicking on the user's name, or you can delete it from the system by clicking the "X" button on the top right in the users bubble.

Add new user to the system

1. Navigate to the Security Administration console and click on the "Users" label in the left panel. You will be presented with the users list.
2. In the header of the right (content) panel, you will see the “Add user” button. Click on it.
3. The panel for user information is presented:

The following information needs to be entered for the user:

- **Username** - this is the user login to the system. Must be unique
- **First Name**
- **Last Name**
- **Initials**
- **e-mail** - the e-mail is required
- **Description** - optional, some short description of the user
- **Password**
- **Disabled** - if checked, created record of the user, but the user is disabled in the system - cannot login or use any of the resources.
- **Roles** - you can select which roles can the user have by selecting them from the dropdown, or clicking on the "Add" button.

Manage Users

Add User

Username:	<input type="text"/>
First Name:	<input type="text"/>
Last Name:	<input type="text"/>
Initials:	<input type="text"/>
e-mail:	<input type="text"/>
Description:	<input type="text"/>
Password:	<input type="password"/>
Repeat Password:	<input type="password"/>
Disabled	<input type="checkbox"/>

Roles

Roles:	none
Select Role:	<input type="button" value="admin ▾"/>



4. Click on the Save button to save the information about the user

Update User information

To update the user information, first navigate to the Security Administration console.

Select "Users" from the left panel and wait for the list in the right panel to load. Once loaded, find the user in the list and click on the name.

You will be presented with the User Edit panel:

Manage Users

Add User

Username: pavle

First Name: Pavle

Last Name: Jonoski

Initials: PJ

e-mail: pavle.jonoski@interworks.co

Description: Pavle Jonoski, some small c

Password:

Repeat Password:

Disabled

Roles

Roles: admin user

Select Role:

Change any information if needed and click "Save". The information about the user will be saved.

Leave the password fields empty to leave the old password unchanged.

Delete a user

Locate the user in the users list in the right panel.

Click on the black X button in the upper right corner of the user bubble.

Confirm the deletion of the user.

Protected Resources Administration

To manage the protected resources, navigate to the Security Administration console and click on the "Web Resources" button in the left panel.

A list of protected Web Resources and URL patterns is presented on the right side.

Add Protected Web Resources

To add new rule for protected Web Resource, click on “Add Protected URL” button.

This will display the panel for adding new protected web resources.

The screenshot shows the 'Manage Web Resources Access' configuration page. The URL Pattern is set to '/testResource/*'. Under 'HTTP Methods allowed', 'post' and 'delete' are removed, and 'GET' is selected. Under 'Allowed Roles', 'Administrator' is selected. Buttons for 'Save' and 'Cancel' are at the bottom right.

You need to enter the following information:

- **URL Pattern** - the URL of the protected resource. You can use wildcard "*", meaning "any character". Thus if you enter "/myResource*" would match any path (stripped from the domain and the context path) that starts with "/myResource".
- **Allowed HTTP Methods** - specify a list of HTTP methods which someone can use to access this resource. You can use "Any" to specify that the resource will be available for any (all) HTTP request types (GET/PUT/POST/DELETE)
- **Allowed roles** - specify which roles can access the protected resource.

Click Save, to save the changes and add the rule for the protected resource on the server.

Delete rule for protected resource

To delete previously entered rule for protected Web Resource, first navigate to the Security Administration console.

Select "Web Resources" from the right panel and wait for the list of web resources to be populated on the right.

Select the Web Resource you want to remove, and click on the button in the upper right corner of the bubble.

Confirm to remove the rule.

Update protected web resource

To update a rule for particular web resource, first first navigate to the Security Administration console.

Select "Web Resources" from the right panel and wait for the list of web resources to be populated on the right.

Find the Web Resource you want to edit, and click on the URL pattern. You will be presented with the panel containing the rule for the protected resource.

Edit the information and click Save.

Server Site

The security API provides the infrastructure and functionality for the security mechanism.

This layer uses the following programmatic services to manage the security Data Model:

- User Services - basic management and administration of users: view, add, remove, update user info; add, remove, update roles
- Consumer Services - management of the OAuth Consumers (apps): view, add, remove Consumer
- Token Services - management of the OAuth tokens (both Request and Access Tokens)
- Protected Web Resources Services - management of the protected Web Resources

These services and the corresponding programmatic model currently reside in **net.megx.megdb** Chon bundle.

The rest of the infrastructure and functionality for the security mechanism is listed bellow

Security API

Security API is a separate sub-module and currently exists as a Chon bundle **net.megx.security.api**.

This bundle provides the following infrastructre to the other bundles and submodules of the Micro B3 IS Web Platform:

- Authentication Handlers
- Authentication Managers
- Security Context and Authentication object
- Basic Authentication Exceptions
- Key-Secret pair generator/provider
- Utilities

Authentication Handlers

Authentication Handler is a provider of Authentication Object. The base purpose of the Authentication Handler is to create Authentication object based on the request parameters, current security context, the resource which is being accessed and the user which is trying to access the resource.

The authentication handler may choose not to create authentication if the input parameters does not satisfy the condition for creation of Authentication (invalid access, authentication already in session etc.).

The Authentication Handler may choose to throw Authentication Exception if something is wrong with the input parameters, thus breaking the security chain and not allowing for the request to pass.

The Security API provides an interface for Web Authentication Handler that builds the authentication according to the current HTTP Request. It defines two types of handlers: Web Login Authentication Handler and OAuth1 Authentication Handler.

Web Login Authentication Handler

Creates Authentication based on the HTTP Request. This handler must provide support for Form Login via HTTP POST and logout functionality. The user credentials are extracted from the HTTP request and the user is checked against the User Database. If everything is fine, the appropriate Authentication is created for the user in question.

The logout functionality clears the current Security Context, effectively destroying the Authentication obtained previously via the form login.

This handler uses two basic endpoints:

- Login endpoint - this is a URL on which the login form is submitted
- Logout endpoint - the URL on which the request for logout is submitted

OAuth1 Authentication Handler

Provides OAuth1 Server authentication/authorization endpoints and request validation.

The handler is registered to the following endpoints:

- Request Token Endpoint - URL on which the OAuth1 client can obtain Request Token
- Authorization Endpoint - URL on which the User (Resource Owner) can authorize the Request Token which in turn will be traded for Access Token on the next step
- Access Token Endpoint - URL on which the client (Consumer) can trade the authorized Request Token for Access Token

Beside these 3 main endpoints, the handler must be configured to process the requests for any Resources that need to be protected with OAuth1. This configuration is done in the Security Filter.

Authentication Manager

The responsibility of the Authentication Manager is to check the authentication against the accessed object and to verify the authorization.

The default provided authentication manager takes into account the Authentication obtained from the current Security Context and the accessed object. The decision for accessing the object is based upon the User's roles and permissions.

The Web Authentication Manager checks the roles from the Authentication created by some of the Authentication Handlers against the matched Web Resource Permission. If no Web Resource Permission (Protected Resource Rules) matches the current request, the request is passed.

The Authentication Manager does the check last in the Security chain of processing to make sure that all Handlers that needed to process the request had already done so.

Security Context and Authentication Object

Security Context is a context stored in the current session (if available) that holds security related information. The basic idea is to store the current Authentication obtained in some previous step in the current session. Also other information about the current state relevant to the authorization/authentication mechanism is stored in this context, such as the last exception, the last requested URL etc.

The Security Context must be available only in the current session. Different sessions must have different contexts.

Authentication Object

The Authentication is a representation of the authentication received from one of the Handlers. This object can be used to confirm that the request currently taking place is done by the user already authenticated.

The Authentication object must be immutable. Changes to the user principal or the roles and permission must not be available and must not take effect to the state of the security context i.e. the sub-sequential requests must see the original Authentication obtained and stored to the Security Context.

Basic Authentication Exception

The Security API provides base security exception which then could be extended to fit the desired granularity of exceptions.

The base exception is AuthenticationException, which could be thrown anytime there is some kind of a problem during Authenticating/Authorizing the user.

Key-Secret Pairs Generator Provider

This sub-module provides service for generating secure-random key and secret pairs. It is part of the crypto package of the Security API.

Utilities

The Security API provides a set of utilities that could be used to interact, manipulate and ease the use of the API.

The following utilities are provided:

- WebUtils - utilities for extracting relevant information from the HTTP Request
- WebContextUTils - provides easy way of managing the Security Context in HTTP Session

Functionality

The security filter is a sub-module that filters the HTTP requests that are send to Micro B3 IS Web Platform. Conceptually it is placed between the client and the Web Platform, intercepting all of the requests and enforcing security checks on the request.

The main functionality of the security filter is to authenticate and authorize specific requests coming to Micro B3 IS Web Platform. Provides:

- OAuth v1 server side mechanism for authentication/authorization on specific resources on the Web Platform
- Standard web login/logout functionality
- Authorization the request for specific resource
- Administration of the users and protected resources

Security Flow

The security filter basically wraps the Web Platform and acts as a separate layer that processes the request before it reaches Chon CMS. For each and every request the following flow is executed:

1. Extract the request path
2. Check if the request is for a static resource (request path matches a static resource such as some PNG image, CSS file or similar)
 1. If it matches - just continue with the filter chain - no extra security checks are performed.
3. Find the security entry-points that match the request path.
 1. Pass the request for processing on the security entry point that matches the path. If more than one entry point matches, the ordering of the entry points is determined by their "order" property.
4. Search the database for protected Web Resources Permissions (Rules for protected resources)
 1. If such rules exist for the current request, pass the processing to the Authentication Manager to decide whether the access is authorized or not

5. If an Authentication has been created or already exist in the current Security Context, populate the HttpServletResponse UserPrincipal with the Authentication
6. Pass the execution to the next in the filter chain - effectively passing the request to Chon

Security Filter module

The security filter is a Chon bundle "**net.megx.security.filter**". It leverages:

- Resources/Templates mechanism provided by Chon to expose its administration part
- OSGI HTTP services to register a filter that intercepts all requests
- REST extension of Chon - for exposing protected REST Services for administration of the security

Security Entry Points

A security entry point is a set of one or more URLs (HTTP endpoints) for which particular Authentication Handler is invoked. For example the URL "http://megx.net/j_security_check" is an endpoint for the standard web login (it's the URL for the submit action for the login form) and its handled by the StandardWebLogin Authentication Handler. In the security filter this URL, along with the logout url are part of the "web_login" security entry point.

You can register as many security entry points as you need with the security filter, thus enabling different mechanisms of obtaining authentication/authorization for different resources.

Configuring the entry points

The entry points are configured in the Security Filter configuration file.

The accepted properties are:

- **name** - string, the name of the security entry point. Should be unique in the security entry point names.
- **class** - string, the fully qualified name of the implementation class for the security entry point. The class must implement `net.megx.security.filter.SecurityFilterEndpoint` interface. The following implementations are available:
 - `net.megx.security.filter.impl.WebLoginSecurity` - wraps the standard Web Login Authentication Handler. Handles form login and logout.
 - `net.megx.security.filter.impl.OAuth_1_Security` - wraps the OAuth v1 Authentication Handler. Handles the OAuth authentication/authorization, provides the necessary URL endpoints for OAuth version 1 and checks the requests signing needed for OAuth v1.
 - `net.megx.security.filter.impl.AnonnimusAuthenticationEndpoint` - provides anonymous Authentication in guest session.

- **urlPattern** - string|Array, the URL pattern or patterns for which this entry point triggers. This may be a single URL pattern or an array of patterns. If an array is specified, the entrypoint is triggered if any of the patterns match the request path. The patterns themselves must be valid Regular Expressions.
- **order** - number, the order of the entry point. If more than one entry point do match the request path, the order of processing is determined by sorting the entry points according to the **order** property - the smaller the value, the larger the priority.
- **enabled** - boolean, whether this entry point is enabled or disabled.

order and **enabled** are optional. The default values are:

- **order** - 0
- **enabled** - true

Working Example

Here is a working example of an entry point - the OAuth version 1 entry point defined in the filter:

```
{  
    name: 'oauth_v1_security',  
    class: 'net.megx.security.filter.impl.OAuth_1_Security',  
    urlPattern: [  
        '/oauth/request_token',  
        '/oauth/access_token',  
        '/oauth/authorize',  
        '/ws/pubmap.*'  
    ],  
    order: 0,  
    enabled: true  
}
```

As seen from the configuration, OAuth v1 security entry point "listens" and is triggered for 4 different URL patterns:

1. /oauth/request_token - the Request Token Endpoint
2. /oauth/access_token - the Access Token Endpoint
3. /oauth/authorize - Authorization URL
4. /ws/pubmap.* - The PubMap RESTfull Web Services. This patterns matches any path that starts with '/ws/pubmap'.

Note that the patterns are not full URLs, but only a path. The Request Path which is matched against these patterns is stripped down from the domain name and the application context path.

Security Filter Configuration

The Security Filter can be configured via JSON configuration file in the bundle itself.

The file is the standard Chon configuration file called `net.megx.security.filter.json` in `__config` directory of the bundle. This file gets copied into `<deployment_dir>/config/` directory upon deployment of the bundle.

The following configuration options are available:

- `path` - string, the path to the templates and static resources of the bundle. This is standard Chon property.
- `filter` - JSON Object, this is the actual configuration for the filter. It has the following options:
 - `enabled` - boolean, whether the filter is enabled. If this is set to `false`, the filter will be initialized and all of the services will be started, but every request that is intercepted by the filter is ignored and passed down the filter chain.
 - `entrypoints` - JSON Array, array of configuration options for the entry points. See the section "Security Entry Points" above for detailed explanation of the configuration options for particular entry point.

Example configuration

Here is an example configuration of the security filter:

```
__config/net.megx.security.filter.json
{
    path: '$resources/net.megx.security.filter',
    filter: {
        enabled: true,
        entrypoints: [
            {
                name: 'standardLogin',
                class: 'net.megx.security.filter.impl.WebLoginSecurity',
                urlPattern: [
                    '/j_security_check.*',
                    '.*admin/logout.do'
                ],
                order: 1
            },
            {
                name: 'oauth_v1_security',
                class: 'net.megx.security.filter.impl.OAuth_1_Security',
                urlPattern: [
                    '/oauth/request_token',
                    '/oauth/access_token',
                    '/oauth/authorize',
                    '/ws/pubmap.*'
                ],
                order: 0
            },
            {
                name: 'anonnymus',
                class:
'net.megx.security.filter.impl.AnonnimusAuthenticationEndpoint',
                urlPattern: '/.*',
                order: 2,
                enabled: false
            }
        ]
    }
}
```