idegaWeb Build Site v.3.0

Project Documentation

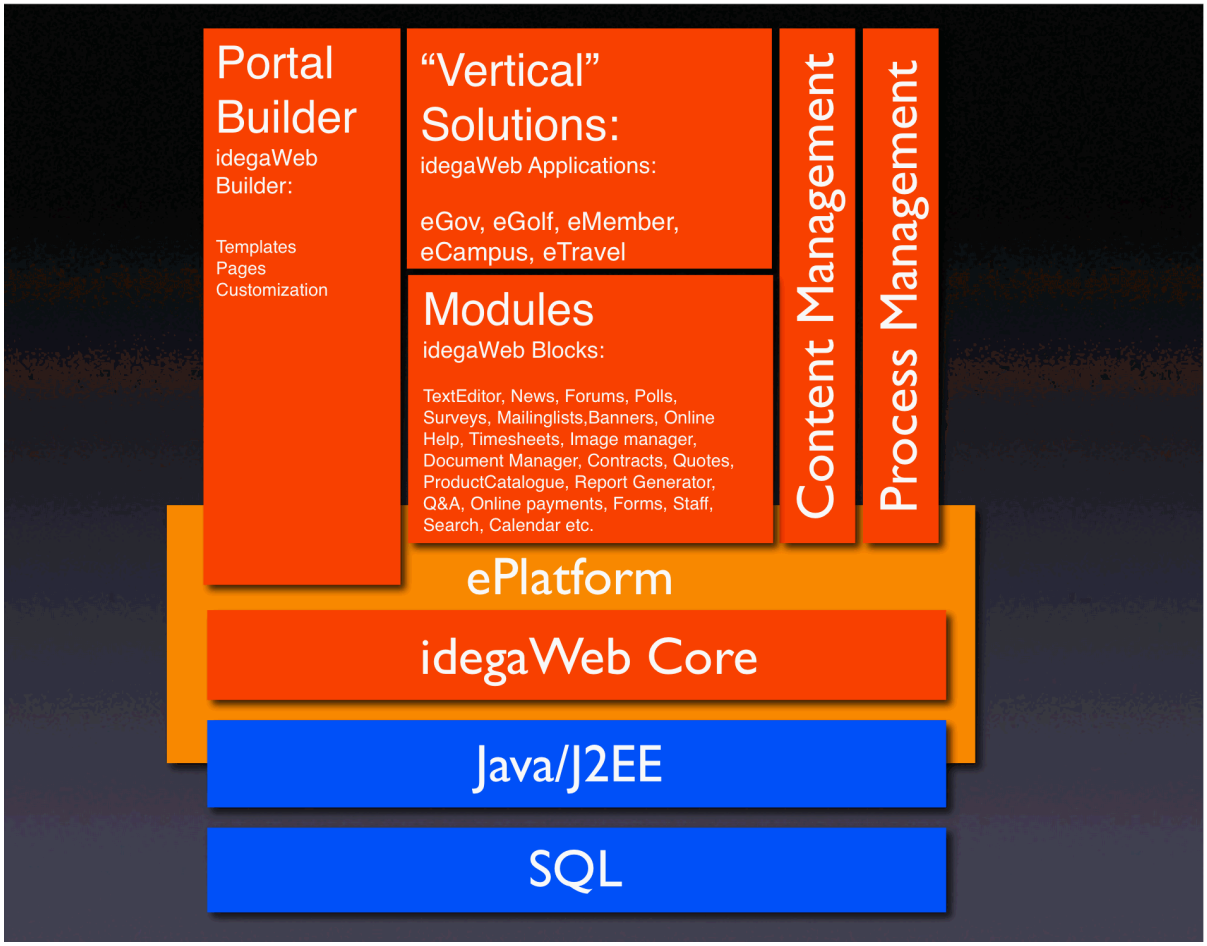
idega software

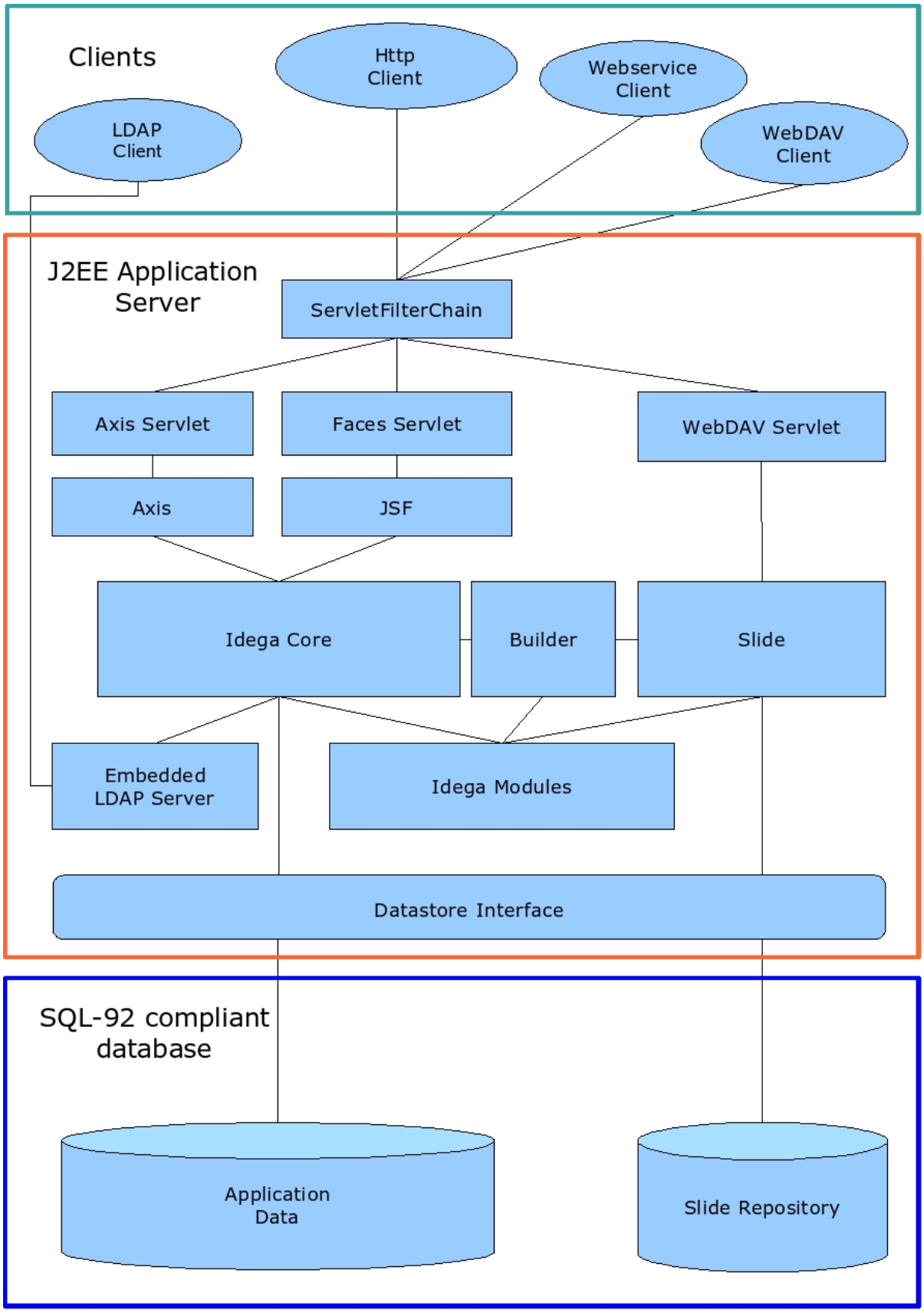
08 December 2006

Table of Contents

1.1 Architectural Overview

Architectural Overview





Design Goals

Open

IdegaWeb is a J2EE application using many open standards like XML avoiding proprietary formats. By integrating exclusively open source frameworks and software idegaWeb keeps itself independent from any vendor. Open means also easy to extend, change and to maintain (e.g., all system data of idegaWeb is human readable).

Multiplatform

IdegaWeb is solely based on Java 2 Enterprise Edition. For deployment only a Java Virtual Machine (1.4+) implementation, a J2EE Application Server or a Servlet Container and a SQL-92 compliant database is needed.

For testing purposes the minimum system requirement is an operating system that has a Java Virtual Machine: In that scenario Apache Tomcat is installed as server and the database is the HSQLDatabase embedded. Apache Tomcat is open source and is also purely written in Java.

Portable

IdegaWeb has not any system-dependent aspects, that means an application can be moved from one system to another.

IdegaWeb even provides tools for importing parts of an idegaWeb application into another idegaWeb application.

Multilingual

Java supports internationalization by using Unicode character encoding and handles local customs.

IdegaWeb localizes all application level user messages and keeps all content localizable.

Modular

- Modules

  An IdegaWeb web application is actually a package composed of between 20-30 modules. The standard platform package has a standard set of modules such as traditional CMS modules (e.g.,Text, Images, Articles, Document Management). Each module has it's User Interface logic, Business and Data or Persistence logic. Modules are also referenced in idegaWeb as bundles.

- Layers

  IdegaWeb is build of several layers each of them only dependent on the underlying layers.

- Plugins

  Several plugins interfaces are implemented allowing to add functionality.

- Service Locators

  Some modules provide service interfaces, that is the concrete implementation is provided by another module.

Flexible

Each web application defines which modules to use and what version. IdegaWeb has a very modularized architecture and most applications are packages composed of between 20-30 modules. IdegaWeb includes a Manager application to install or to update those modules.

Accessible

IdegaWeb provides different interfaces to communicate with clients and other applications like Servlets, web services, WebDAV and LDAP.

Reusable

IdegaWeb separates content and customized functionality from implementation by assembling web applications through pages. A page defines the layout and what modules are shown. Further a page contains all configuration data and customization for the modules on that page.

Modules are never written for a certain web application, but they are customized in pages that belong to a certain web application.

Secure

IdegaWeb supports different login methods like normal login, basic authentication and single sign on. IdegaWeb provides a multi-user, role-based access to resources of a web application.

Simple

All administrator tools for maintaining and creating a web application are fully integrated in the system. Database tables are automatically created and modified by the system. No additional applications or tools need to be installed.

Important Frameworks

JavaServer Faces (implementation by MyFaces)

JSF is a server-side user interface component framework. JSF is J2EE standard and includes a set of APIs for representing user interface (UI) components and managing their state, handling events and input validation, defining page navigation, and supporting internationalization and accessibility. In contains further a default set of UI components and two JavaServer Pages (JSP) custom tag libraries for expressing a JavaServer Faces interface within a JSP page. JSF defines a very clear processing request life cycle using the server-side event model and state management.

Since version 3.0 the MyFaces JSF engine renders all User Interface.

See: JavaServer Faces Technology

Apache Axis

Framework implementing SOAP enabling web services.

See: WebServices -Axis

Jakarta Slide

Jakarta Slide is a content repository which serves as a basis for the idegaWeb content management system, providing full WebDAV support.

See: Jakarta Slide -The Jakarta Slide project

Java LDAP Server

Providing a LDAP server.

The project was stopped at sourceforge JavaLDAP Server Project but forked and moved to codehaus Codehaus: LDAPServer

JDom

Framework for reading and writing XML documents.

See: JDOM

HSQL Database

HSQLDB is the leading SQL relational database engine written in Java. It has a JDBC driver and supports a rich subset of ANSI-92 SQL. Since it is based on Java it can run together with idegaWeb and is therefore the default test database for the system.

See: HSQLDB

Batik

Providing a SVG Renderer, Images are rendered to PNG or JPEG to the client.

See: Batik SVK Toolkit

POI

Framework to handle MS-Office formats.

See: Jakarta POI - Java API To Access Microsoft Format Files

iText

Framework to handle PDF formats.

See: iText, a Free Java-PDF Library

JasperReports

Reporting tool delivering documents in PDF, HTML, XLS and XML files.

See: JasperForge

ICU4J

International Components for Unicode, providing advanced libraries for Unicode support.

See: ICU

COS

Providing servlet support classes.

See: com.oreilly.servlet

JUG

Framework for generating 128-bit Universally Unique Identifiers.

See: Java Uuid Generator

Apache Lucene

Lucene is a high-performance, full-featured text search engine library.

See: Lucene

Ehcache

Ehcache is a widely used Java distributed cache for general purpose caching.

See: Ehcache

Junit

JUnit is a simple framework to write repeatable tests.

See: JUnit, Testing Resources for Extreme Programming

DWR

DWR Direct Web Remoting allows Javascript in a browser to interact with Java on a server working like a bridge from Java to Javascript and back. Primarily used for creating Ajax like GUIs in idegaWeb.

See: Direct Web Remoting

JAI

JAI Java Advanced Imaging provides a set of object-oriented interfaces that support a simple, high-level programming model.

See: Java Advanced Imaging

1.2 Modules and Components

Modules and Components

Application, Layers, Modules and Components

An idegaWeb application is a composition of many modules. A module is a piece of software that handles a certain aspect. Most modules are quite small. Typically each module has it's User Interface logic, Business and Data or Persistence logic but some modules implement just some functionalities that do not have a user interface at all.

Modules in idegaWeb are usually synchronous with bundles and often are also called Blocks.

Modules can be assigned to layers. Modules of one layer are usually depending on some of the modules of underlying layers. In that way an application is made up of several layers.

1.2.1 UI Components

UI Components

Components of the user interface can mainly be categorized into the groups PresentationObjects, PresentationObjectContainer, Blocks and JSF UIComponents.

PresentationObjects

Components that directly extend PresentationObject are rather simple elements like Text and Inputs. They do not normally have any children.

Note that from version 3 PresentationObjects are also JSF UIComponents therefore they could technically contain children. In Version 2 and before they can not contain any children.

PresentationObjectContainers (extending PresentationObject)

Components that extend PresentationObjectContainer are more complicated elements. They can contain children, that is other components can be added to that component. A very good example of this kind of class is the Table/Table2 component.

Blocks (extending PresentationObjectContainer)

Components that extend Blocks are rather sophisticated components. Blocks provide usually interactive and database connected functionality. Like PresentationObjectContainers they can contain children. Blocks provide an encapsulated functionality by putting and linking together several PresentationObjects and PresentationObjectContainers. In that way items or features are available as Blocks ready for use. They can simply be added to a website in the idegaWeb Builder. Thus it is not necessary to assemble or rebuild the same feature again and again. Furthermore Blocks also implement caching facilities.

Good examples of Blocks are Login, Forum, ImageGallery and News.

Caching of Blocks

Blocks are cached and stored with a unique cachekey which is just a String. This String is by default made out of the classname of the Block, the permission rights the viewer has and the locale he is using the module in. Therefore caching for Blocks that are stateless is covered by default.

Every cacheable block must call the method setCacheable() in its constructor. Additonally a method getCacheKey() that returns a unique prefix for the key must be implemented.

If a block can have different states the method getCacheState() of the super class Block must be overridden. This method composes and returns a string that depends on the state of the block. In other words the string corresponds to the state of the block.

JSF UIComponents

From version 3.0 the platform supports standard JSF UIComponents the same way as older style

PresentationObjects.

1.2.1.1 UI Component Table

Component Table

The UI component table is represented by two classes in idegaWeb. Either com.idega.presentation.Table or com.idega.presentation.Table2 can be used. Table2 is a better and newer class than Table and usage of this class is therefore recommended.

The Table2 component is more clearly structured and renders out a more semantically correct html.

Below an example of usage of Table2 taken from com.idega.block.school.presentation.SchoolSeasonEditor is shown.

The class SchoolSeasonEditor is an editor for editing school seasons. A list of all already defined school seasons is shown. The user can edit, delete or create seasons.

First the table is created and cellpadding and cellspacing is set to zero. The width of the table is set to 100%. A style class is assigned to the table.

```
Table2 table = new Table2();
table.setCellpadding(0);
table.setCellspacing(0);
table.setWidth("100%");
table.setStyleClass(STYLENAME_LIST_TABLE);
```

Now a columngroup is defined. Five columns without a specified width and the last two columns with a specified width.

```
TableColumnGroup columnGroup = table.createColumnGroup();
TableColumn column = columnGroup.createColumn();
column.setSpan(5);
column = columnGroup.createColumn();
column.setSpan(2);
column.setWidth("12");
```

Next the header of the table is defined and the cells are filled. The header title are the localized names of "Name", "Category", "Start", "End" and "Due date". The last two columns have no headers because the first will contain a link for editing the season and the second will contain a link for deleting the season.

```
TableRowGroup group = table.createHeaderRowGroup();
TableRow row = group.createRow();
TableCell2 cell = row.createHeaderCell();
cell.setStyleClass("firstColumn");
cell.add(new Text(localize("name", "Name")));
row.createHeaderCell().add(new Text(localize("category", "Category")));
row.createHeaderCell().add(new Text(localize("start", "Start")));
```

```
row.createHeaderCell().add(new Text(localize("end", "End")));
row.createHeaderCell().add(new Text(localize("due_date", "Due date")));
row.createHeaderCell().add(Text.getNonBrakingSpace());
cell = row.createHeaderCell();
cell.setStyleClass("lastColumn");
cell.add(Text.getNonBrakingSpace());
```

Next the body of the table is defined.

```
group = table.createBodyRowGroup();
```

Next the body is filled using an iterator over all seasons.

```
int iRow = 1;
java.util.Iterator iter = seasons.iterator();
while (iter.hasNext()) {
        SchoolSeason season = (SchoolSeason) iter.next();
        row = group.createRow();

        try {

                ...fetch the category, start date, end date and due date

                ...define the variable edit as Link for editing the SchoolSeason of
the current row

                ...define the variable delete as another Link for deleting the
SchoolSeason of the current row

                cell = row.createCell();
                cell.setStyleClass("firstColumn");
                cell.add(new Text(season.getSchoolSeasonName()));
                row.createCell().add(new Text(category != null ?
localize(category.getLocalizedKey(), category.getName()) : "-"));
                row.createCell().add(new Text(startDate != null ?
startDate.getLocaleDate(iwc.getCurrentLocale(), IWTimestamp.SHORT) : "-"));
                row.createCell().add(new Text(endDate != null ?
endDate.getLocaleDate(iwc.getCurrentLocale(), IWTimestamp.SHORT) : "-"));
                row.createCell().add(new Text(dueDate != null ?
dueDate.getLocaleDate(iwc.getCurrentLocale(), IWTimestamp.SHORT) : "-"));
                row.createCell().add(edit);
                cell = row.createCell();
                cell.setStyleClass("lastColumn");
                cell.add(delete);

                if (iRow % 2 == 0) {
                        row.setStyleClass(STYLENAME_LIST_TABLE_EVEN_ROW);
                }
                else {
                        row.setStyleClass(STYLENAME_LIST_TABLE_ODD_ROW);
                }
        }
        catch (Exception ex) {
```

```
                ex.printStackTrace();
        }
        iRow++;
}
```

Finally building the Table2 object is finished. The table can now be added to the page.

1.2.1.2 UI Component Link

Component Link

The UI component link is represented by com.idega.presentation.Link.

Below an example of usage of Link taken from com.idega.block.school.presentation.SchoolSeasonEditor is shown.

The class SchoolSeasonEditor is an editor for editing school seasons. A list of all already defined school seasons is shown. The user can edit, delete or create seasons.

A link using an Image is created. Next two parameters are added to the link.

```
Link edit = new Link(getEditIcon(localize("edit", "Edit")));
edit.addParameter(PARAMETER_SCHOOL_SEASON_PK, season.getPrimaryKey().toString());
edit.addParameter(PARAMETER_ACTION, ACTION_EDIT);
```

Another option without using an Image is

```
Link edit = new Link(new Text(localize("edit", "Edit")));
```

A simple Link referring to a URL looks like

```
Link idega = new Link("Idega", "http://www.idega.is/");
```

A mail reference is done like

```
Link sendEmail = new Link(email.getEmailAddress(), "mailto:" +
email.getEmailAddress())
```

1.2.1.3 UI Component Image

Component Image

The UI component image is represented by com.idega.presentation.Image.

Below an example of usage of Image taken from com.idega.block.school.presentation.SchoolBlock is shown.

The class SchoolSeasonEditor is an editor for editing school seasons. A list of all already defined school seasons is shown. The user can edit, delete or create seasons.

The variable iwb holds the DefaultIWBundle of the bundle com.idega.block.school. The path "shared/edit.gif" points to a gif image that is located in the folder /resources/shared/ of the bundle.

```
/**
 * Returns the default edit icon with the tooltip specified.
 * @param toolTip The tooltip to display on mouse over.
 * @return Image The edit icon.
 */
protected Image getEditIcon(String toolTip) {
        Image editImage = this.iwb.getImage("shared/edit.gif", 12, 12);
        editImage.setToolTip(toolTip);
        return editImage;
}
```

The methods in DefaultIWBundle look like

```
public Image getImage(String urlInBundle, int width, int height)
{
        return getImage(urlInBundle, "", width, height);
}

public Image getImage(String urlInBundle, String name, int width, int height)
{
        return new Image(getResourcesURL() + slash + urlInBundle, name, width,
height);
}
```

Finally the URL is composed to "*application context*/idegaweb/bundles/com.idega.block.school.bundle/resources/shared/edit.gif" where *application context* depends on the installation of the web application. The returned image has the width and height of 12 pixel and is not localized since it is located in the shared folder of the bundle.

Tool Tip

The method setToolTip() inherited from PresentationObject sets the title attribute. The tooltip is shown on mouse over.

Alt Text

The method setAlt(String) sets the alternative text of the image. This text is shown when the image could not be loaded. If this method is not called the name of the image is taken as alternative text. In the example above the alternative text is empty but the tooltip is set.

1.2.1.4 UI Component Form

Component Form

The component form is represented by com.idega.presentation.ui.Form.

Below an example of usage of Form taken from com.idega.block.school.presentation.SchoolBlock is shown.

The class SchoolSeasonEditor is an editor for editing school seasons. A list of all already defined school seasons is shown. The user can edit, delete or create seasons.

The Form is created. By default

- attribute "action" is set to send to the page itself
- attribute "method" is set to "post"
- attribute "target" is not set

```
Form form = new Form();
form.setStyleClass(STYLENAME_SCHOOL_FORM);

...some code to define the table...

form.add(table);
form.add(new Break());

SubmitButton newLink =
(SubmitButton) getButton(new SubmitButton(localize("season.new", "New season"),
PARAMETER_ACTION, String.valueOf(ACTION_NEW)));
form.add(newLink);

add(form);
```

It is important to understand that all elements that should be submitted must be included in the form by adding them to the form. In the example above the table might contain some input elements. Therefore the table is added to the form and also the submit button.

Finally the form is added to the page. In the example above the method add(PresentationObject) of PresentationObjectContainer is used.

If a page is desired as target there is a convenient method called setPageToSubmitTo(ICPage) that can be used.

```
public void setPageToSubmitTo(ICPage page) {
        setPageToSubmitTo(((Integer) page.getPrimaryKey()).intValue());
}
```

1.2.1.5 UI Component Text

Component Text

The component text is represented by com.idega.presentation.text.Text.

Below an example of usage of Text taken from com.idega.block.school.presentation.SchoolBlock is shown.

The class SchoolSeasonEditor is an editor for editing school seasons. A list of all already defined school seasons is shown. The user can edit, delete or create seasons.

Text is a quite simple element as shown below.

```
...some code...

TableCell2 cell = row.createHeaderCell();

...some code...

cell.add(new Text(localize("name", "Name")));

row.createHeaderCell().add(new Text(localize("category", "Category")));
row.createHeaderCell().add(new Text(localize("start", "Start")));
row.createHeaderCell().add(new Text(localize("end", "End")));
row.createHeaderCell().add(new Text(localize("due_date", "Due date")));
row.createHeaderCell().add(Text.getNonBrakingSpace());

...some code...

cell.add(Text.getNonBrakingSpace());

...some code...
```

The method localize(String, String) is defined in SchoolBlock and looks like

```
public String localize(String textKey, String defaultText) {
        if (this.iwrb == null) {
                return defaultText;
        }
        return this.iwrb.getLocalizedString(textKey, defaultText);
}
```

The variable iwrb holds the IWResourceBundle of the bundle com.idega.block.school. This method tries to retrieve the localized version of the text defaultText by using the key textKey. If either iwrb is not set or the key could not be found it returns the string defaultText.

Non breaking space

The static method getNonBrakingSpace()

```

```

```
Text.getNonBrakingSpace();
```

is misspelled and sets a non breaking space that is it sets " ".

Break

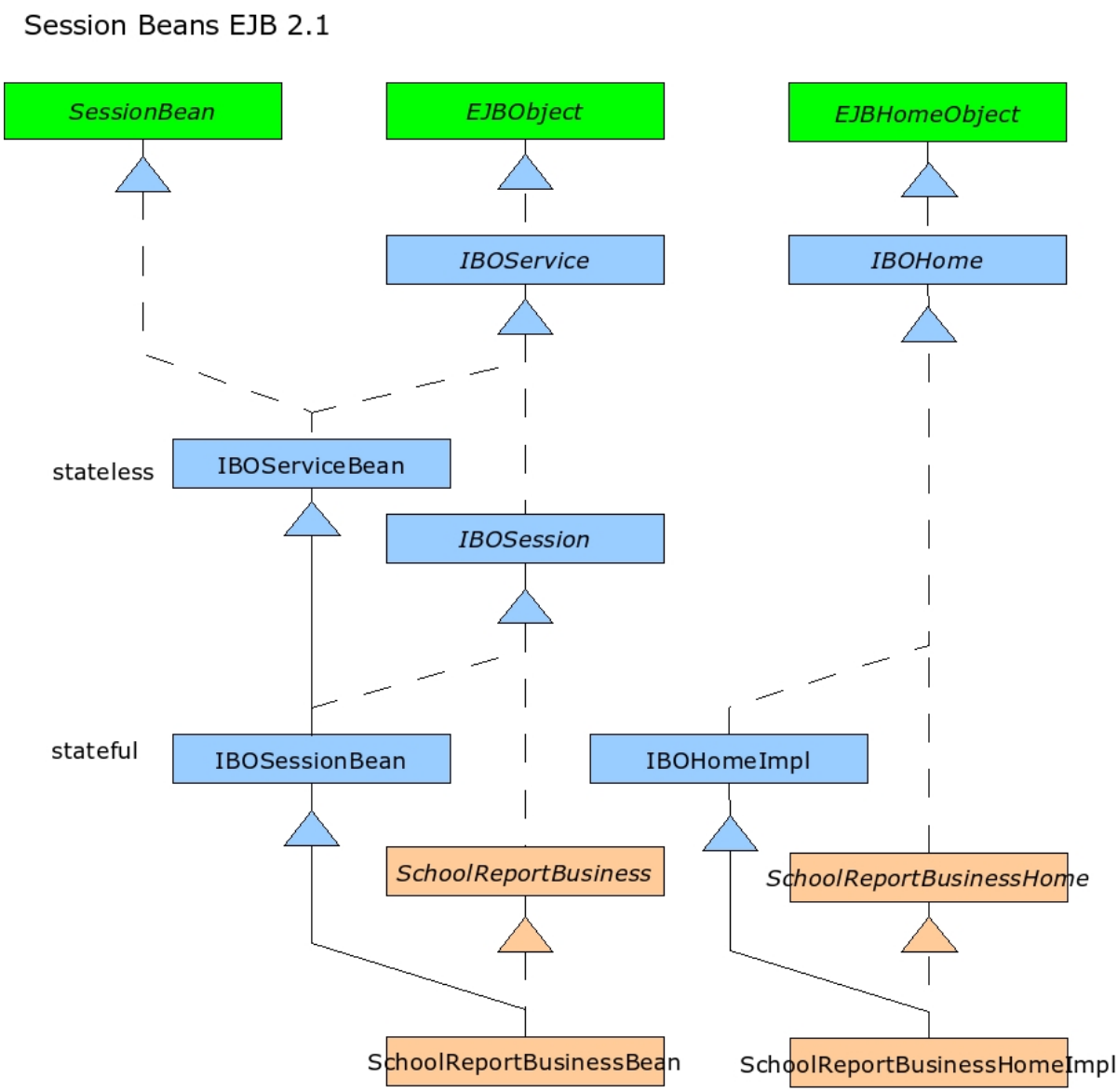
The static method getBreak()

```
Text.getBreak();
```

sets a break that is it sets "
".

1.2.2 Business Logic Components

Business Logic Components



IdegaWeb builds on an EJB 2.1 (Enterprise Java Beans) compatible architecture for objects in the Business Logic tier.

The picture above shows how stateful and stateless session beans are implemented in idegaWeb.

SchoolReportBusiness serves as an example for a stateful session bean. SchoolReportBusiness is the bean's remote interface extending the IBOSession interface thus extending javax.ejb.EJBObject. SchoolReportBusinessBean is implementing both the bean's remote interface SchoolReportBusiness and javax.ejb.SessionBean by extending IBOSessionBean.

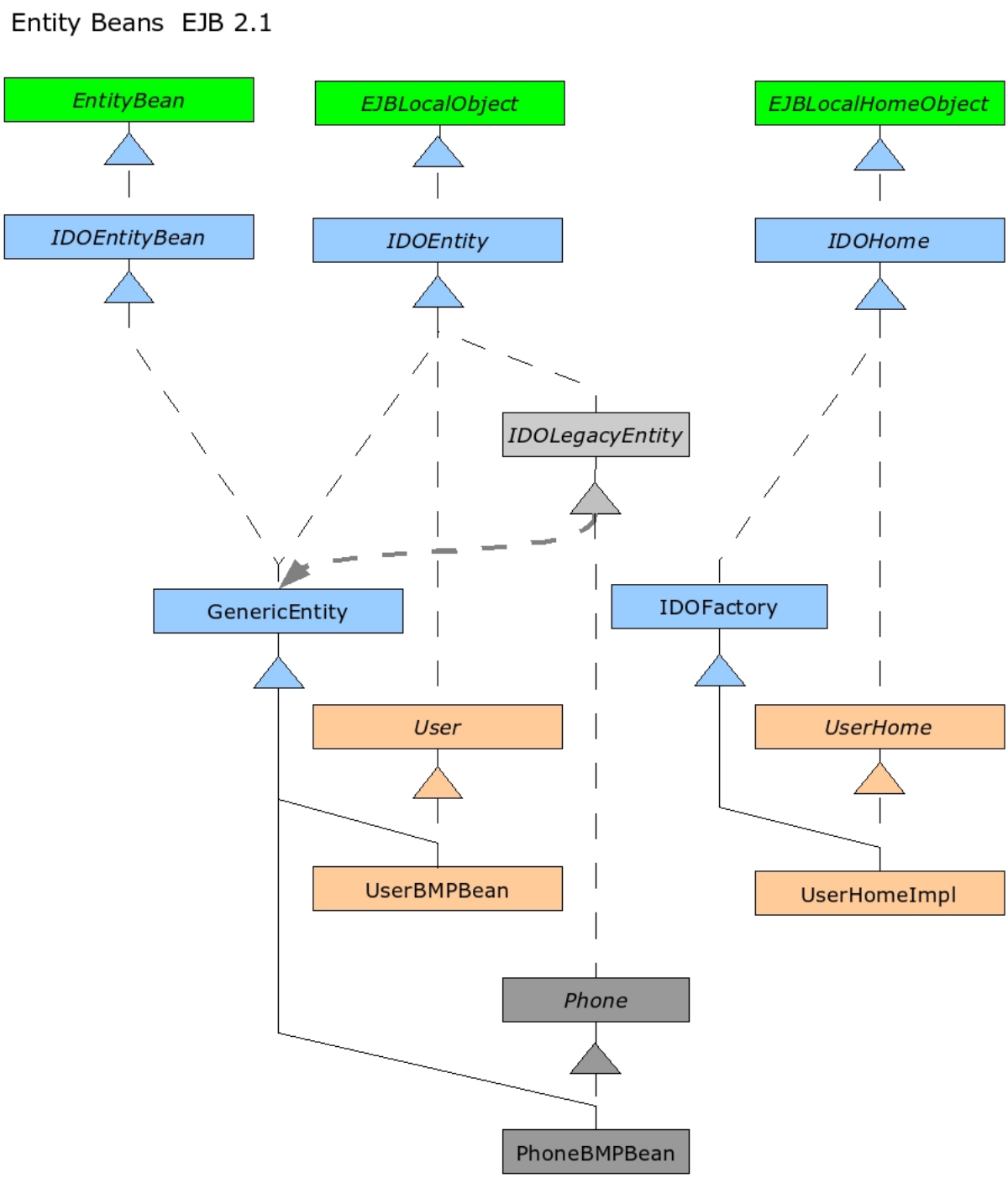
If SchoolReportBusiness was a stateless session bean, it is sufficient that SchoolReportBusiness extends IBOService and SchoolReportBusinessBean extends IBOServiceBean. SchoolReportBusinessBean has in this case also to implement the remote interface SchoolReportBusiness.

The bean's remote home interface is SchoolReportBusinessHome. It extends IBOHome thus extending javax.ejb.EJBHomeObject. SchoolReportBusinessHome is implemented by SchoolReportBusinessHomeImpl that extends IBOHomeImpl. IBOHomeImpl implements IBOHome.

For looking up session beans in idegaWeb IBOLookup is used.

1.2.3 Persistence Components

Persistence Components



IdegaWeb builds on an EJB 2.1 (Enterprise Java Beans) compatible architecture for objects in the Persistence tier.

The picture above shows how entity beans are implemented in idegaWeb. There are not any remote interfaces but local ones. Entity beans are not remote accessible in idegaWeb. Clients modify entities indirectly by calling business methods of the corresponding session beans.

The entity User is an example how an entity bean is implemented.

The bean's local interface User extends IDOEntity, in that way it extends javax.ejb.EJBLocalObject. UserBMPBean is implementing both the bean's local interface User and javax.ejb.EntityBean by extending GenericEntity.

The bean's local home interface is UserHome. It extends IDOHome thus extending javax.ejb.EJBLocalHomeObject. UserHome is implemented by UserHomeImpl that extends IDOFactory. IDOFactory implements IDOHome.

IDOLegacyEntity is a deprecated interface that some entities are still extending. IDOLegacyEntity extends IDOEntity. Although not declared GenericEntity implements all methods of IDOLegacyEntity. Usage of IDOLegacyEntity is meant to be phased out and was introduced in a refactoring in the whole platform where also the more modern IDOEntity was also introduced.

The entity Phone serves as an example: The bean's interface Phone extends IDOLegacyEntity. PhoneBMPBean extends GenericEntity and implements both the bean's local interface Phone and javax.ejb.EntityBean.

PhoneHome and PhoneHomeImpl is not shown in the picture, they are implemented in the same way like UserHome and UserHomeImpl.

For looking up entity beans in idegaWeb IDOLookup is used.

1.2.4 Bundles

Bundles

Modules in idegaWeb are also called bundles in the system.

Beside java packages or classes every bundle has some special folders containing all resources or configuration files for each module.

Properties of a Component - properties Folder

Every bundle has a folder named "properties". This folder contains a file "bundle.pxml". The ending might be a little bit misleading as the file is just an xml file. The bundle.pxml file provides information about the components of the bundle and the name of the module.

Every component of the bundle has its own property file. The bundle.pxml file maps the name of the component to the corresponding component property file (e.g., the component "com.idega.block.text.presentation.TextReader" is mapped to the "com.idega.block.text.presentation.TextReader.pxml" component property file).

The component property file contains information about the type of the component (e.g., "iw.element", "iw.data", "iw.block", "iw.plugin.user") and the corresponding properties (e.g., a component of the type "iw.element" has properties specifying available settings in the idegaWeb Builder for the corresponding component).

A simple example follows:

The bundle "com.idega.core" has a folder "properties". This folder contains a file "bundle.pxml". There is an entry for a subclass Text of PresentationObject with the full name "com.idega.presentation.text.Text". This component Text has the component property file "com.idega.presentation.text.Text.pxml". This file says that Text is an "iw.element". The type "iw.element" means it is a presentation object that is available in the idegaWeb Builder. The component property files defines further that there are set methods for the text to be displayed (e.g., Text can be set to show "Hello world") and the style of the displayed text (e.g.,Text can be set to show the string "Hello world" with the style "bold").

Resources - resources Folder

Another folder in a bundle is called "resources". It contains all kind of files that the bundle needs to reference on a direct URL to work (e.g., image files). Especially important are the files that are needed for internationalization. See chapter "Internationalization".

Web Application Deployment Descriptor - WEB-INF/web.xml File

If there are classes in a bundle that request entries in the web.xml file of the web application those bundles should provide a folder WEB-INF with a file named web.xml. The necessary entries should be put into this web.xml file.

The web.xml file of a bundle can be actually considered as part of the whole web.xml file. All web.xml

files of the several bundles are merged into one web.xml file when the web application is build into a war file with the idegaWeb maven plugin (the goal "iw-application:war" of the plugin should be called).

See also paragraph "Creating a Webarchive File For Distribution" in chapter "Installation".

Database Tables of a Component: ICObject and ICObjectInstance

ICObject

The previous chapter described how the properties of a component are stored in the file system. There is a corresponding database table to the component property files called IC_OBJECT. The IC_OBJECT table is just representing the ICObject entity. Every ICObject belongs to a certain component. Its attributes are

- identifier (an incrementing number)
- name of the component (e.g., "Text")
- name of the implementing class (e.g., "com.idega.presentation.text.Text")
- type (e.g., "iw.element")
- bundle to that the component belongs (e.g., "com.idega.core")

The values are written to the database table when deploying a new web application with an empty database by reading the component property files during startup. Entries can be changed or added with the developer tool BundleComponentManager.

ICObjectInstance

When a component is put on a idegaWeb Builder page this use of the component gets an entry into a special table IC_OBJECT_INSTANCE. Again this table represents just the ICObjectInstance entity. The attributes are

- identifier (an incrementing number)
- reference to the ICObject (e.g., reference to the "Text" ICObject)
- reference to the page where the component is placed (e.g., reference to the page "pages/overview")

The ICObjectInstance is used to link other objects to a certain component in a certain page. Note that a component is often used more than one time in different pages or in the same page. A simple reference to the class of the component was ambiguous.

Loading of a Bundles

Loading of a bundle starts with finding and reading the bundle.pxml file of the bundle. This is done in two different ways:

- Loading of Bundles: Platform 2

  When an idegaWeb application starts it looks for bundles located in the special /idegaweb/bundles folder. Every bundle that belongs to the application has an own directory there. Each directory is looked up for the bundle.pxml property file that was mentioned above.

- Loading of Bundles: Platform 3.5 (unreleased as of October 2006)

  In idegaWeb platform 3 so called JarLoaders check all jar files located in in the /WEB_INF/lib

folder. One of those JarLoaders is IWBundleLoader. IWBundleLoader checks if the jar file contains a bundle.pxml file to classify as an idegaWeb bundle.

From the bundle.pxml file the bundle identifier is read. The next step is to create an instance of DefaultIWBundle using the found bundle identifier as a parameter. This instance of DefaultIWBundle represents the corresponding bundle. The object that represents henceforth the bundle is put into a map of IWMainApplication. In that way the object DefaultIWBundle can be accessed later by calling corresponding methods of IWMainApplication.

During initialization of the instance of DefaultIWBundle all components defined in the bundle property file are written to the database if not already present. In other words the database table IC_OBJECT is synchronized with the bundle property file.

Creation and Update of Database Tables

In a further step of the initialization all components of the bundle with component type "iw.data" are checked if their corresponding database table is already present in the database or need to be created or updated. Those components are apart from some very rare exceptions entities beans.

If an entity is not defined in the bundle property creation of the database table is not done during startup even if the database table is not present. The corresponding database table of the entity bean is in that case created not until an instance of the entity is used or created the very first time. That means that the database table might be created when the web application is running in regular operating mode causing delays arising in request responses. This behaviour is not desired therefore it is highly recommended to register all entity beans of a bundle in the corresponding bundle property file.

IWBundleStartable and IWBundleStarter

After creating the DefaultIWBundle some special start methods of the bundle are finally called. Those start methods are defined in special classes that are implementing the interface IWBundleStartable. The interface IWBundleStartable defines two methods

- start(IWBundle)
- stop(IWBundle)

The start method should be called when the bundle is added to the web application or during startup. The stop method should be called when the bundle is removed from the web application or during shutdown.

There are two ways to define which implementors of IWBundleStartable should be used for a certain bundle.

- IWBundleStarter

  If a class named IWBundleStarter exists in the package that corresponds to the name of the bundle this class is used as an IWBundleStartable (e.g., the bundle com.idega.block.forum has the class com.idega.block.forum.IWBundleStarter). The class must implement IWBundleStartable.

  If this class exists this IWBundleStartable is always called first.

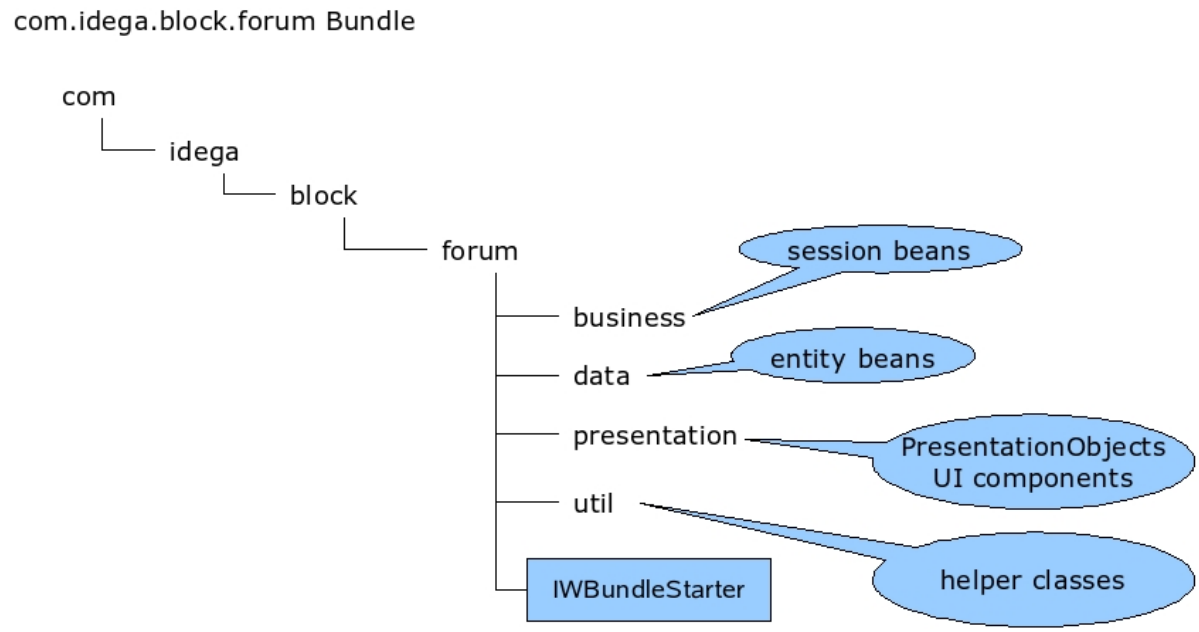
- IWBundleStartable entry in the bundle property file

  An IWBundleStartable can be defined in the bundle property file under the key "iw_bundle_starter_class". The defined class must implement IWBundleStartable.

If this property exists this IWBundleStartable is called after a call of IWBundleStarter if that one exists.

Note that neither an IWBundleStarter nor an IWBundleStartable property entry is necessary. There could be only an IWBundleStarter, only an IWBundleStartable property entry or none of both.

Package Structure of a Bundle



The picture above shows a typical package structure of a bundle. There are four package component names that appear in almost every bundle. The structure follows the Model-View-Controller (MVC) design pattern.

- business

  This is the package where all session beans of the bundle are placed. It should contain all business relevant classes. It contains all classes of the bundle that belong to the controller of the MVC pattern.

- data

  This is the package for all entity beans of the bundle. It might also contain POJO (plain old Java objects) data objects. It contains all classes of the bundle that belong to the model of the MVC pattern.

- presentation

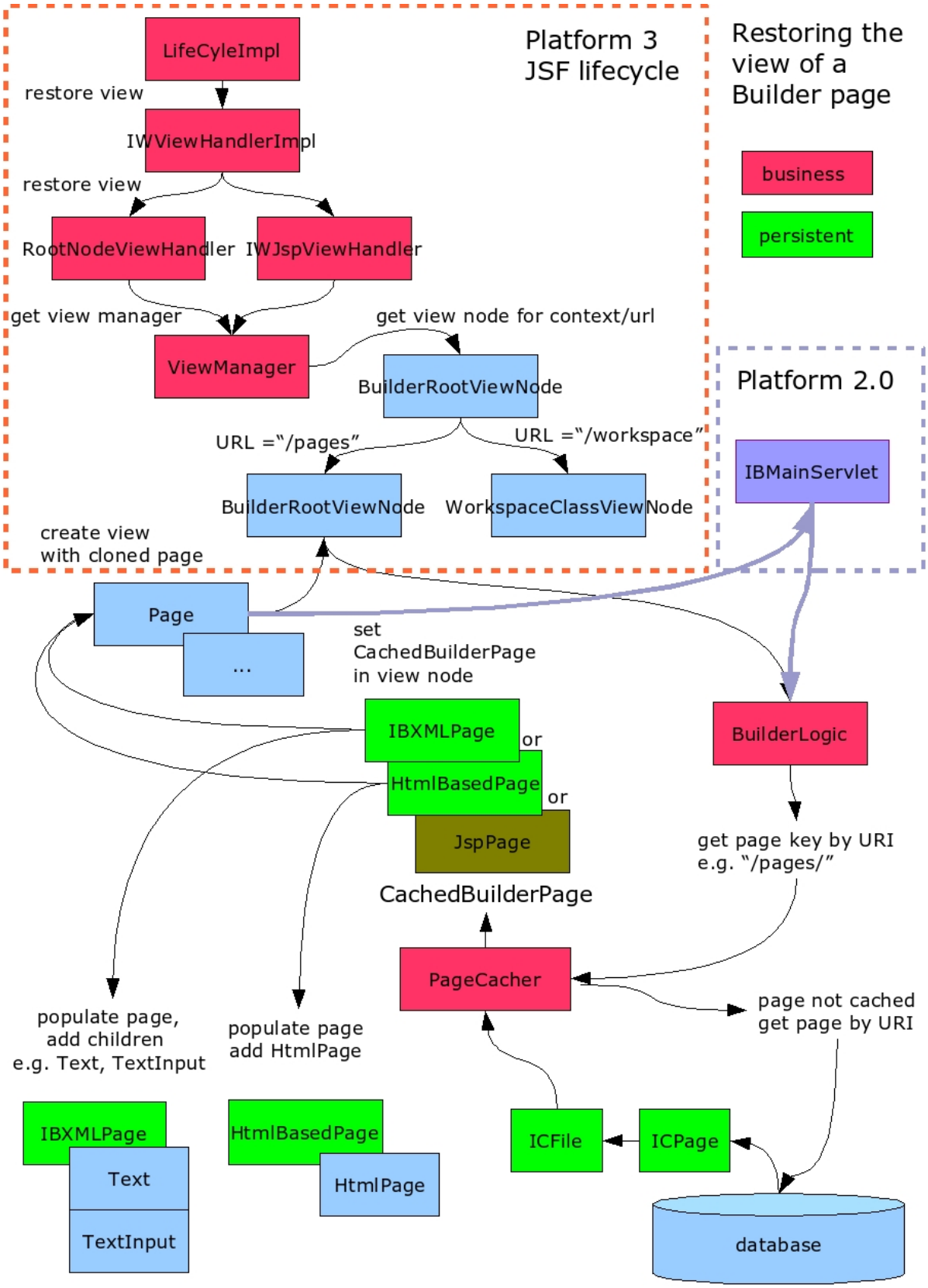
  This package contains PresentationObjects, PresentationObjectContainers, Blocks and further classes of the bundle that belong to the user interface. This package contains all classes of the bundle that belong to the view of the MCV pattern.

- util

  The util package contains util classes that are used by classes in the packages above. Those classes are in most cases quite small and have a technical focus. They must not implement business logic. They must neither be persistent nor belong to the user interface.

1.3 Request Processing

Request Processing



Difference between Platform 2 and 3

The greatest difference between these versions is in the implementation of the rendering engine.

In platform 3 the rendering engine has been replaced with a JSF one (more specifically the MyFaces implementation). Alongside this a large effort has also been put into providing near perfect backwards compatibility with all older pre version 3 style components so they can run as before in this new rendering engine.

The reason why this is done is to make the platform grow out from a proprietary architecture into a more standardised, feature-rich and future-proof foundation.

Platform 3: Request Processing

When a page is requested by a client the first phase of the JSF life cycle is restoring the view. IdegaWeb has its own view handler implementation called IWViewHandlerImpl. The request could ask for a JSP page, an idegaWeb Builder page or some other page programmed up purely in java code.

For JSP pages and idegaWeb Builder pages, that are stored as JSP pages, the IWViewHandlerImpl uses the IWJspViewHandler. For all other pages, in particular idegaWeb Builder pages stored as IBXML files, the RootViewNodeHandler is taken. Both handlers invoke the ViewManager for getting a view node. The concrete class for the view node depends on the URL. For idegaWeb Builder pages the URL starts with "/page" and a BuilderRootNodeViewNode is used. For other pages than idegaWeb Builder pages the view node can be very different (e.g., for the URL "/workspace" a WorkspaceClassViewNode is retrieved).

Requesting a non-idegaWeb Builder page

When a non-Builder page is requested the ViewManager tries to look up the view node by the URL. In most cases the corresponding view node and URL registration is hard coded and not read from some configuration files. The content of the view node could be defined by classes or by an underlying JSP page.

Requesting an idegaWeb Builder page

To fill an idegaWeb Builder page node with its content the corresponding Builder page has to be looked up. This is done by figuring out the Builder page URL from the request and looking up the Builder page in a special cache called PageCacher. If the page could not be found the ICPage object is fetched from the database by the specified URL. The ICPage object knows the page key and by this page key the content of a page, stored in an ICFile object, is loaded from the database. The content of a page is either a xml document representing an IBXMLPage, a HTMLpage HtmlBasedPage or a JSP page JSPPage.

All three kinds of page classes have logic for handling and rendering their specific document format and the content of the desired page. If the page was just loaded from the database there is no component tree. The loaded page is stored in the cache managed by the PageCacher.

The next step is creating and instantiating the component tree of the page. This process is called populating a page and triggered when a view of this page is needed. The IBXMLPage is populated by all components that are listed in the underlying xml document. The HtmlBasedPage gets just one child called HtmlPage basically containing the HTMLcontent as string. The JSPPage is not populated but written down to the disk on a temporary URL and the request dispatched to that URL. It is then handled

further solely by the JSP engine and JSF as any other JSF/JSP page.

After the last step the stored page in the cache, namely a IBXMLPage or a HtmlBasedPage, is populated. When a view is requested for a view node a clone of the page is made. The page has to be cloned since more than one client could request the same page. A page is usually not stateless but contains dynamic modules that can have different states. Cloning a page means that all its contained modules have to be cloned too. In other words a deep copy of the object is done. For idegaWeb modules the implementation of the clone method is therefore crucial. For regular JSF UIComponents the "clone" method is not used, rather a new instance of the component is created for each request. The properties of this new instance are set by the defined ones in the IBXMLPage.

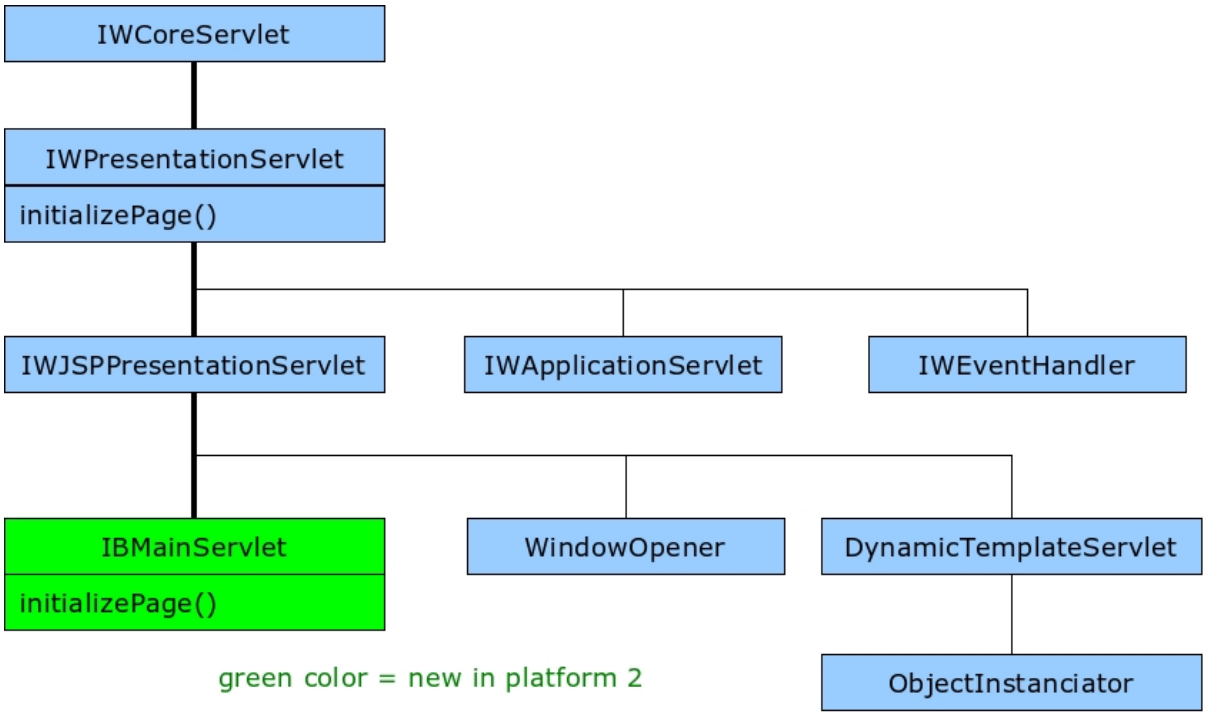
Finally the view node is linked to the cloned page. The life of that page instance depends now on the JSF life cycle.

Platform 2: Request Processing

Requests in platform 2 are not handled by JSF at all. Instead several different idegaWeb servlets handle requests. For requests asking for an idegaWeb Builder page the IBMainServlet is responsible. In platform 2 the URL looks different to the one that is used in platform 3: For Builder pages the BuilderLogic does not figure out the page URI from URL but rather the page id. By this page id the corresponding page can be found in the database. The rest of the procedure is quite similar apart from the special treatment of JSF components that are unknown in platform 2. Furthermore the cloned page is not linked to a view node - there are not any view nodes in platform 2 - but returned to the IBMainServlet.

The picture below shows an overview of the servlets used before platform 3. IdegaWeb Builder pages were introduced with platform 2. From that time the IBMainServlet is fetching pages from the database. Without the IBMainServlet pages were just hard coded. Note that the method initializePage() of IWPresentationServlet is overwritten by IBMainServlet.

There are other servlets used in platform 2 apart from IBMainServlet. These servlets are either extended by JSP pages or mapped on their own URL (e.g., the WindowOpener servlet is mapped on "/servlet/WindowOpener").



1.3.1 Main and Print

Processing and Rendering an idegaWeb page

Main Phase and Print Phase

When creating the response of a request there are two important steps in idegaWeb that are processed after the page has been restored.

They are called main and print methods. In JSF the main() method is similar to the processDecodes() method. The print() method is similar to the encodeBegin(), encodeChildren() and encodeEnd() methods.

First the main() methods of all modules, that is PresentationObjects, that are contained in the restored page are called. The nodes of the module tree or component tree are visited in a recursive depth-first way. The purpose of the main() method is to change the status of the modules depending on the request (e.g., when a form was submitted the modules are processing the submitted values).

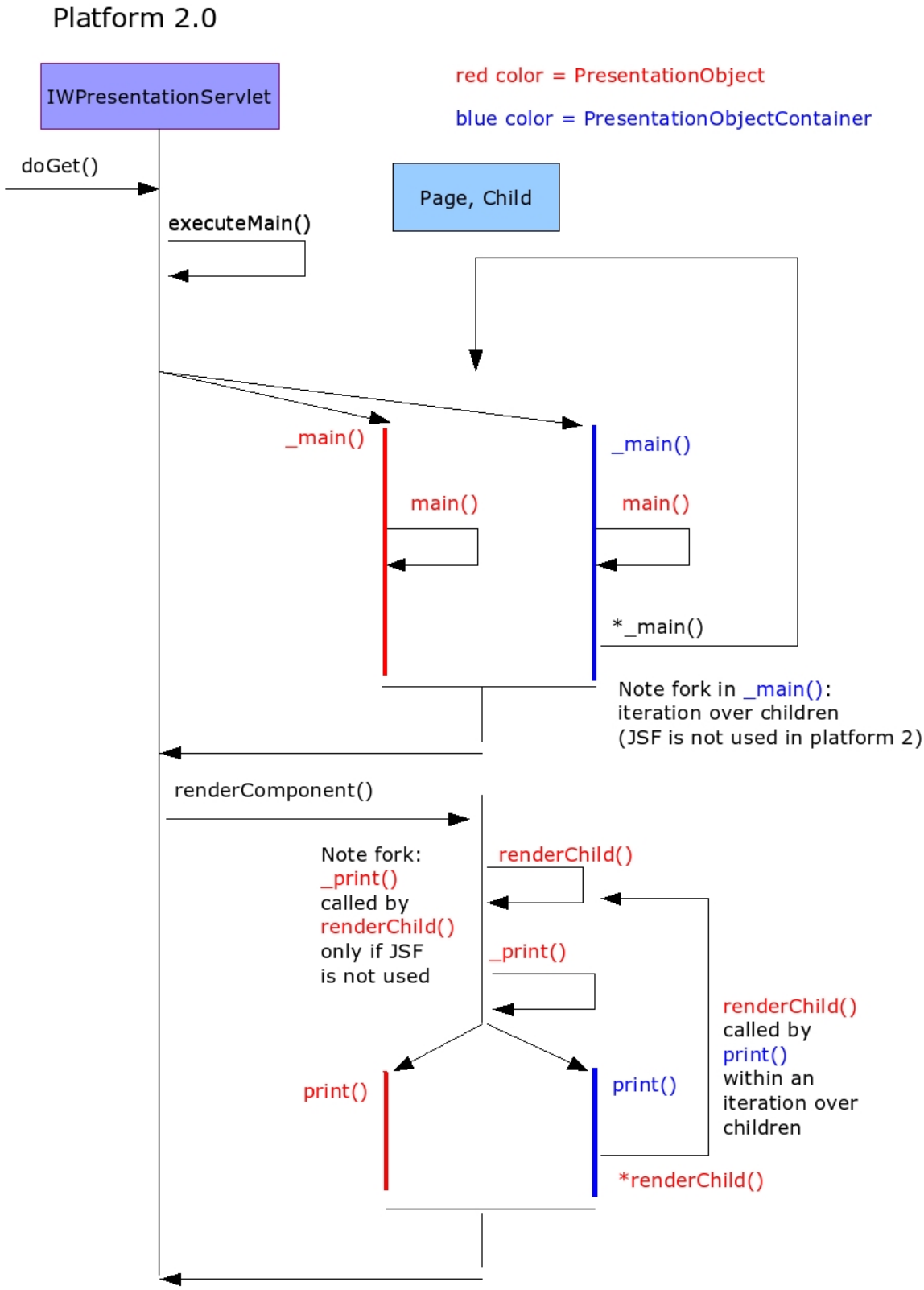
Now the whole component tree has been notified about the request. All components have changed its status if necessary. The process of the request is now finished. Note that modules might depend on each other, that is the process must be processed from all modules first before the rendering of the response can be started.

The second phase is calling the print() methods of all modules. This is also done in a recursive depth-first way. The print() methods are rendering the output accordant to the desired format (e.g., html).

By calling the print() methods recursive depth-first modules containing other modules do not need to know what kind of modules are they containing. For example a table starts rendering itself but then calls the print() methods of all modules that are contained in its cells. After returning from the children of a table the rendering of the table itself is finished.

Both phases are divided into two sub phases. The main phase calls first the _main() method, then the main() method. Likewise the print() phase calls first the _print() method then the print() method. There is only one iteration loop through all children per phase. The _main() method calls the main() method and for PresentationObjectContainer it then calls this on all children components. The _print() method calls the print() method and again for PresentationObjectContainer it then calls this on all children components.

Platform 2: Call of main() and print()



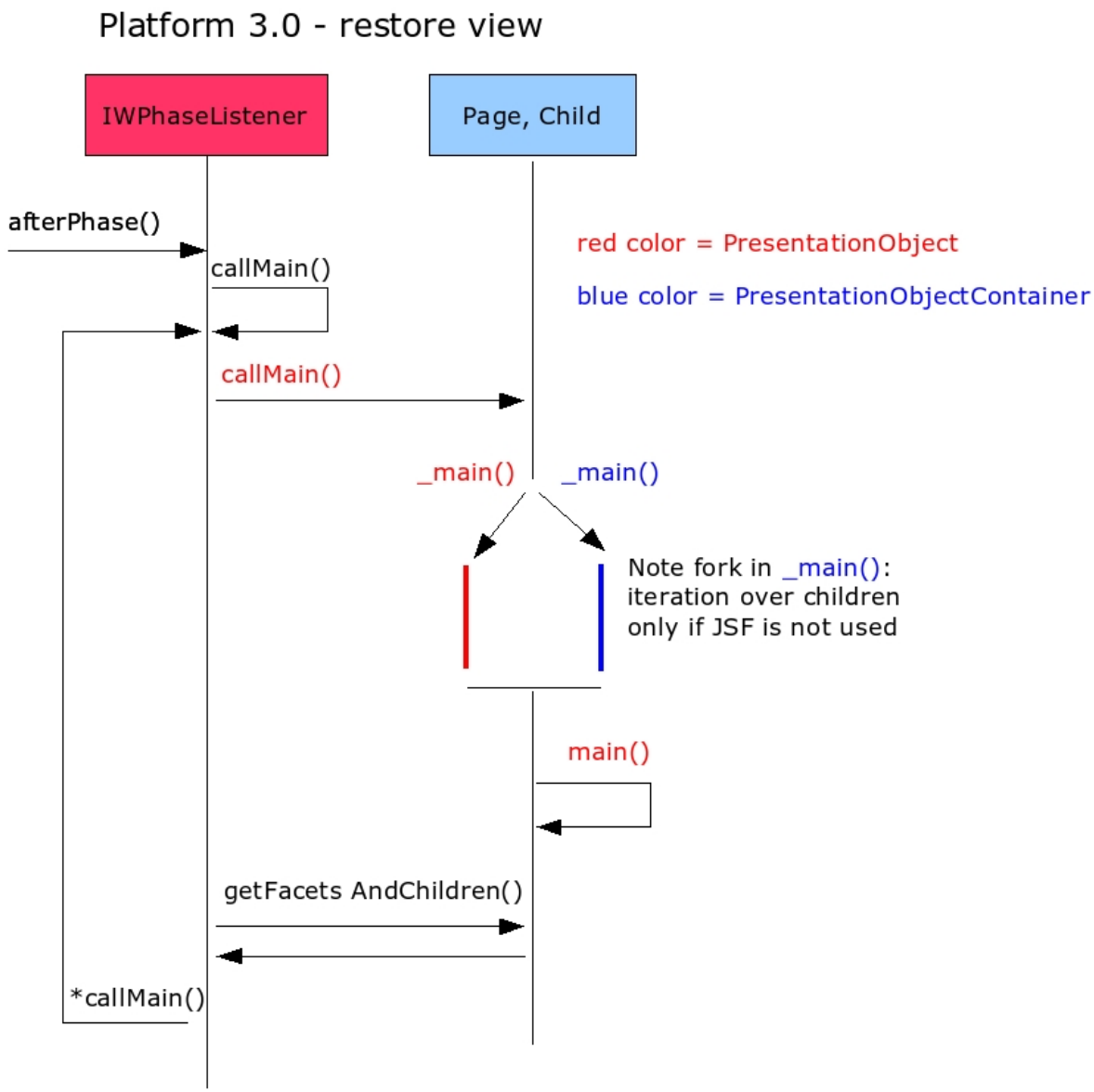
The picture above shows a very simplified sequence diagram of handling a request in platform 2. PresentationObject is a super class of PresentationObjectContainer. PresentationObjectContainers do

have children whereas PresentationObjects do not have children.

Note that PresentationObject only implements the methods _main() and print() but not main() and _print(). Furthermore both implemented methods are not calling the corresponding super methods.

Blocks that are subclasses of PresentationObjectContainer have a special case. That means that subclasses of the class Block should not implement a print() or _print() method. Because of this only the class Block itself implements a print method that is final and can therefore not be overwritten.

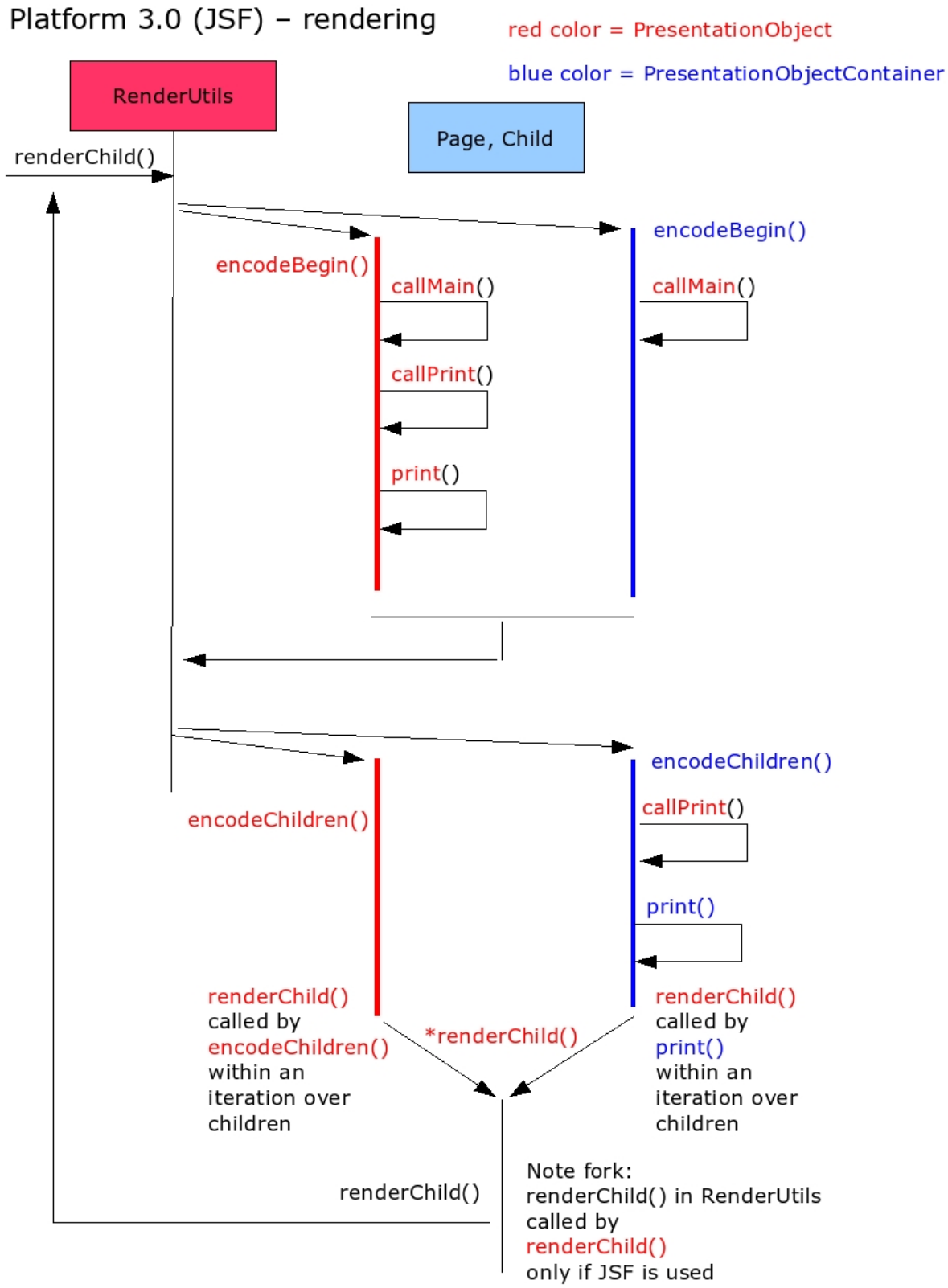
Platform 3: Call of main() during restore view phase



The picture above shows a simplified sequence diagram of the main() method calls after the restore view phase of JSF. The IWPhaseListener is registered into the JSF life cycle by attaching to the restore view phase.

The afterPhase() method of the IWPhaseListener is called after JSF has finished the restore view phase. In JSF all UIComponents can be considered as containers, that is they can have children. For this reason iterating through the children is always done by the IWPhaseListener.

Platform 3: Call of main() and print() during rendering phase



The picture shows a simplified sequence diagram of the main and print method calls during the rendering phase of JSF. The renderChild() method of RenderUtils is involved by the view handler.

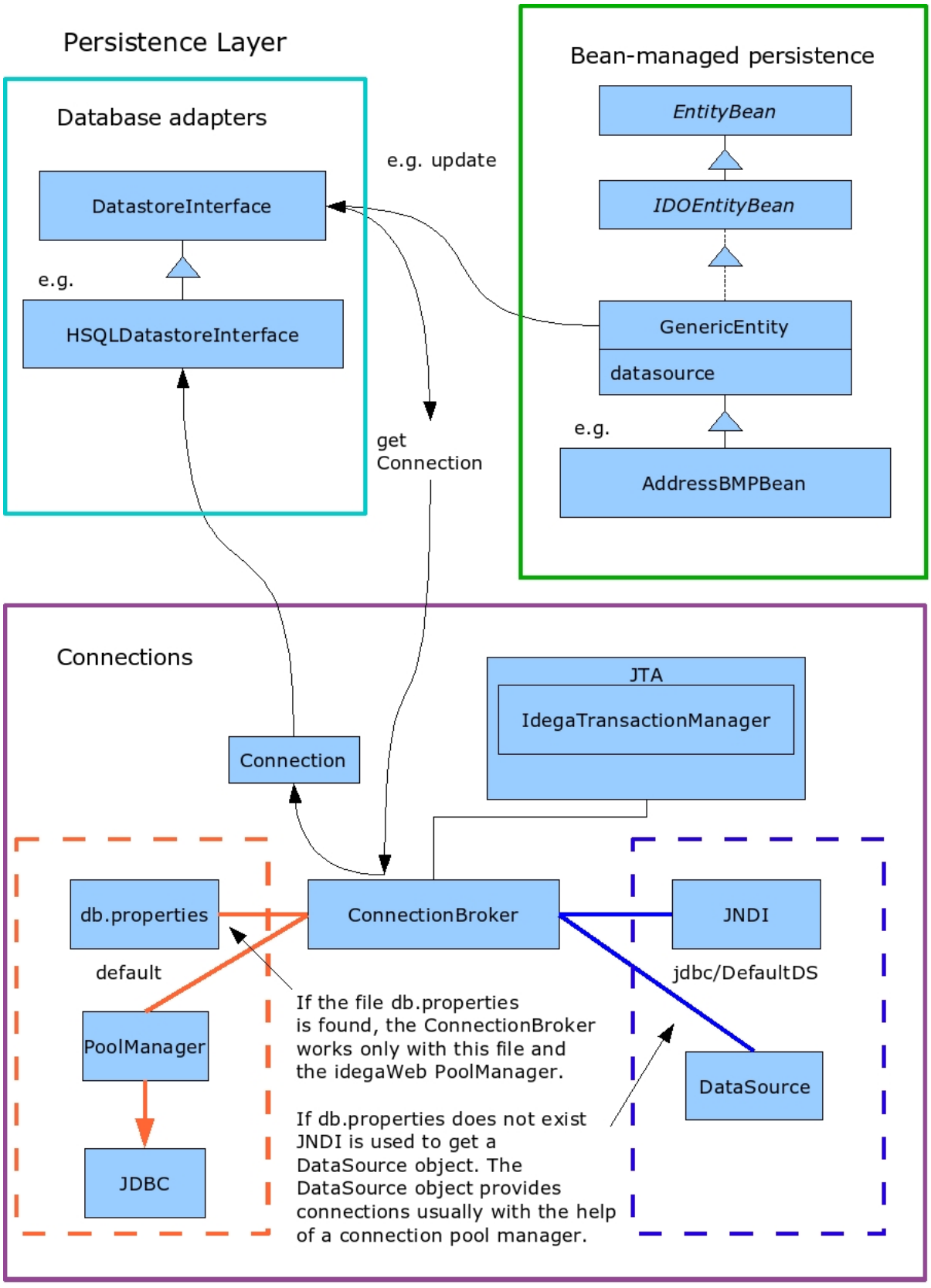
The reason for calling _main() again during the rendering phase is that JSP pages are created and rendered parallel in contrast to the JSF life cycle where the component tree is first created completely, then the request is processed by the components and finally the components are rendered.

In other words the IWPhaseListener is calling the _main() method at the right time but the problem is, that the component tree is not complete yet when the requested page is a JSP page. To be more precise this problem only appears when the JSP page is requested by the client the very first time. When JSF receives the first request for the page, the component tree for the view represented by the JSP page does not exist. JSF creates a tree with just a UIViewRoot at the top and forwards the request to the JSP page to add the real components. The JSP container processes the page, asking the components to render itself. After the very first request henceforth the component tree is restored completely by JSF during the restore view phase and the IWPhaseListener is actually calling the _main() methods of all components that belong to the requested JSP page.

Again when a JSP page is requested the very first time by a client, the component tree is not complete and for this reason the IWPhaseListener is not calling the _main() methods of the missing components. Therefore in the case of a JSP page requested the very first time the main() method has to be called when the components are actually created, that is during the rendering phase.

1.4 Persistence Layer

Persistence Layer



Object-relational Mapping

Entities in idegaWeb are represented by so called "BMPBeans", bean-managed persistence beans, that is the persistent state of the object is handled by the bean itself. BMPBeans are compatible to Enterprise Java Beans EJB implementing the EntityBean interface.

The BMPBean, usually a subclass of GenericEntity, contains all the piece of information for creating the corresponding database tables including relationships to other entities, that is to other BMPBeans. This description is done on a high level independent from a certain database.

Database Adapters

The link to the database is given by a concrete implementations of the abstract interface DatastoreInterface that work as an adapter to the database. In that way differences between databases are compensated. At the same time by using special adapters execution of SQL statements are optimized. idegaWeb supports many databases among them are Oracle, Microsoft SQL Server 2000, Informix Dynamic Server, Firebird Interbase, SapDB and MySQL.

Creation of Tables

Creating and maintaining of database tables is done automatically when the application is started. Usually only one table is created or updated per entity (e.g., if an attribute was added). A special case are many-to-many relationships to other entities. In that case middle tables are also automatically created. This feature of automatically creating and maintaining database tables plays an important role in idegaWeb abilities of self installing the whole web application and later adding or updating of modules.

Managing Connections

idegaWeb can handle more than one database at the same time. At startup the application looks for a certain property file, the db.properties file, for getting information how to connect to databases. If this property file is not found JNDI is used for looking up JDBC standard DataSources.

The default database is found by the data source name "default" in db.properties or by the URL "jdbc/DefaultDS" using JNDI.

After startup the ConnectionBroker knows how to look up different databases by a specified data source name. If a BMPBean wants to store or get its state from the database a connection is fetched by the ConnectionBroker depending on the data source name. If the db.property file is used the idegaWeb PoolManager is called to get a connection. If JNDI is used the returned DataSource Object is called for getting a connection. Usually the connection is taken from an underlying PoolManager handled by the DataSource object.

The right database adapter is assigned by asking the connection for the underlying database.

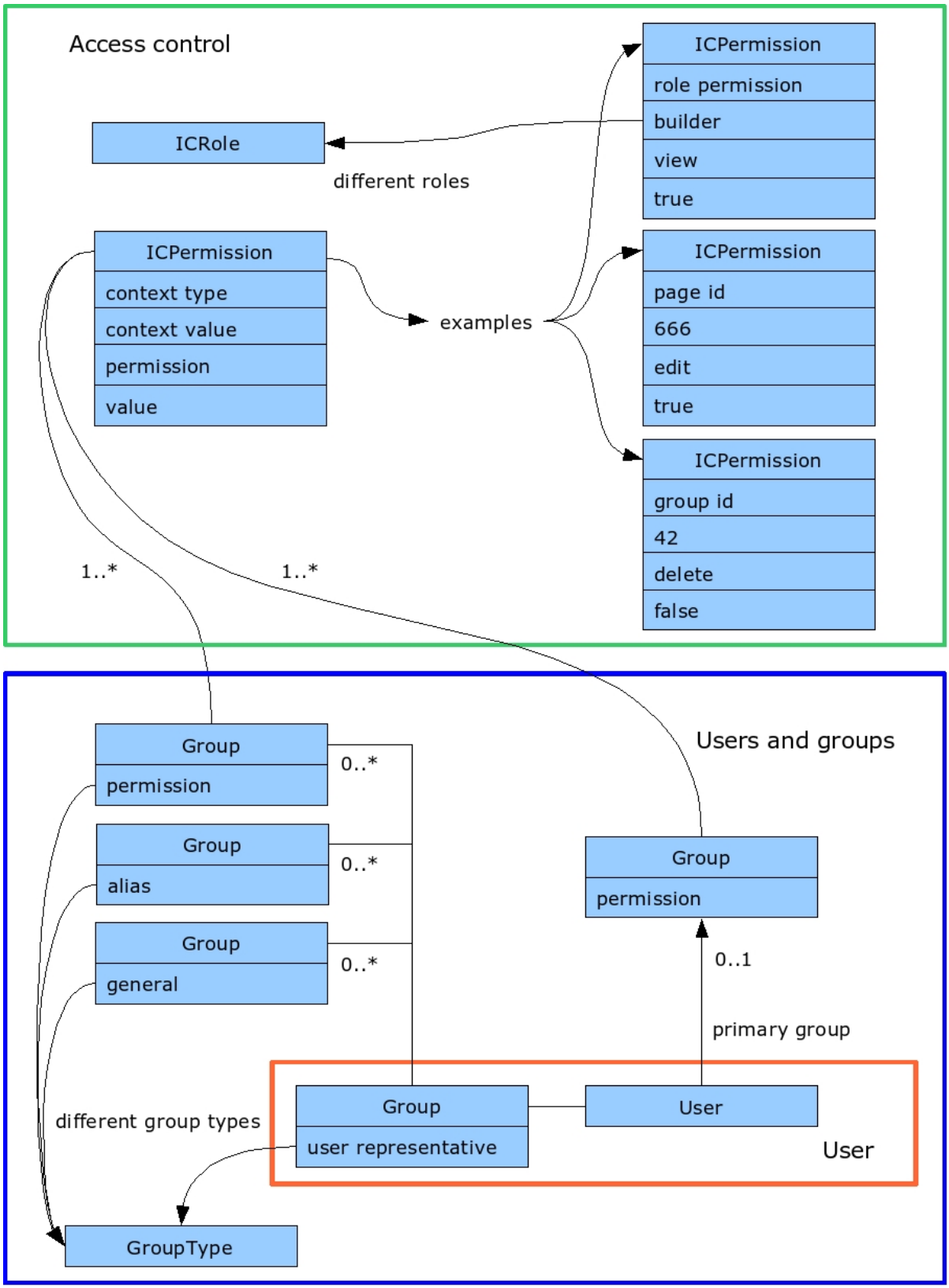
Transactions

If a transaction is started the IdegaTransactionManager reserves the connection gotten from the ConnectionBroker by associating it to the current thread. That means that the connection is not available

for other threads. After finishing the transaction the connection is either committed or rolled-back. After that it is released and returned to the connection pool.

1.5 Users and Groups

User and Groups



Every user object in idegaWeb is associated with a group object, or in other words, every user is also a group at the same time. If a user has a login the user usually belongs to one or more groups. Users and

groups map the real world organization on the system. Most of the user objects represent real persons, but some of some rather represent certain roles like secretary of a company or principal of a school.

Primary Group

The user object points to a special group, called the primary group. The primary group assigns the user to a basis group like citizens or school administrators. Usually the primary group does not change during lifetime of the user object. Nevertheless having a primary group is not mandatory.

Group Types

Every group has a type, in most idegaWeb systems there are four types used: user representative, general, alias and permission:

- user representative

  only used to mark the group object that is associated with the user object.

- general

  mostly used for real world groups to categorize users.

- alias

  an alias group points to another group (e.g., alias groups are applied when mapping non-hierarchical organization charts to a hierarchical tree).

- permission

  marks a group that has certain permissions like administrators, citizens, developers and so on. Permission groups are an older way of handling roles and is the only group type to be visible in the idegaWeb Builder access control system.

  In the future the model is changing making permission groups obsolete: Any group type except "alias" can have permissions or a role.

Apart from the listed ones above there could be other group types. In general group types are used for customizing behaviour for different types of groups.

Permissions

Permission groups have permissions. Permissions control what user and group data a group can view, manage and control. Controlling data means setting permissions.

Important attributes of a permission are:

- context type

  the object the permission refers to like a page, a group or a role.

- context value

  the identifier of the referred object like a page id, a group id or the name of a role.

- permission

  the aspect of the permission like edit, view, delete, add, remove or view.

- value
  the value of the aspect that is true or false.

Note: Unfortunately the meanings of context value and permission are confused in some classes.

Roles

Roles can be considered as job descriptions like builder, schools administrator or content editor. They are a kind of special groups defined by job duties.

Roles are on a higher level than permissions. They are not specific to any data but rather to application functionality or modules.

User Properties

Every user in idegaWeb can keep some special personal settings (e.g., how he would like to get messages).

Those properties are stored in a special user property file for each user. Such a file is created not until a property of a user needs to be stored.

The property file of a user is shared by all modules. Properties are usually grouped. The name of the group corresponds either to the name of the module that sets the properties or to a certain aspect.

By grouping name conflicts among user property names are avoided and the meanings of the properties are much clearer.

Below there is a simple example of a user property file. It says that the user gets email but does not prefer to get emails into the inbox of its user home page in the web application.

The property file is located on the file system and is not stored in the database. See also paragraph "Directory Layout Of an idegaWeb Application" in chapter "Installation".

```
<?xml version="1.0" encoding="UTF-8"?>
<pxml>
  <map>
    <key>
      <name>citizen_account_properties</name>
      <value>
        <map>
          <key>
            <name>msg_send_box</name>
            <type>java.lang.Boolean</type>
            <value>false</value>
          </key>
        </map>
      </value>
      <type>map</type>
    </key>
    <key>
      <name>message_properties</name>
      <value>
        <map>
          <key>
```

```
            <name>msg_send_email</name>
            <type>java.lang.Boolean</type>
            <value>true</value>
          </key>
        </map>
      </value>
      <type>map</type>
    </key>
  </map>
</pxml>
```

1.6 Plugins

Plugins in the User System

Plugins in the User System

The plugin interface UserGroupPluginBusiness declares methods for customization of the behaviour of the user system.

Registering a Plugin

Active plugins are represented by the entity UserGroupPlugIn. Important attributes of UserGroupPlugIn are

- name of the plugin
  This is just a name that describes the functionality.
- plugin type
  Valid values are "group" and "user". The plugin type says where the plugin should work. The plugins with type "group" are only working if the current focus is on a group. The plugins with type "user" are working if the current focus is on a user.
- reference to an ICObject
  The referenced component has to be a plugin (e.g., a reference to the ICObject "is.idega.block.nationalregister.NationalRegisterBusiness").

Every class implementing a plugin for the user system has to implement the UserGroupPluginBusiness interface.

This class (also called component) must be registered as "iw.user.plugin", that is the component has to have the type "iw.user.plugin". This type is an entry in the corresponding component property file of the component. (e.g., the class "is.idega.block.nationalregister.NationalRegisterBusiness" has a type entry with value "iw.user.plugin" in the component property file "is.idega.block.nationalregister.NationalRegisterBusiness.pxml". This file is in the folder "properties" of the bundle "is.idega.block.nationalregister").

If the entry is missing it can be created by using the developer tool BundleComponentManager.

The next step is adding a certain entry into a database table. To do so some database tables have to be viewed.

- First the identifier of the corresponding ICObject has to be figured out by viewing the table ic_object. (e.g., executing the sql statement: "select ic_object_id from ic_object where class_name = 'is.idega.block.nationalregister.NationalRegisterBusiness' " returns the identifier)
- Next step is creating a new instance of the entity UserGroupPlugIn. To create a new instance of UserGroupPlugIn at least the attributes reference to an ICObject and the type has to be set. The reference is set by using the found identifier from the last step. The type depends on the plugin either

the plugin works with groups or with users. In the former case the type is set to "group" in the latter case to "group". The table name of the entity UserGroupPlugIn is "ic_user_plugin". (e.g., assume the returned identifier of the mentioned SQL statement on ic_object above is "42". Executing the SQL statement "insert into ic_user_plugin (plug_in_type, business_ic_object) values ('user', 42)" creates the entry for the "is.idega.block.nationalregister.NationalRegisterBusiness" plugin).

1.6.1 User Type Plugins

User Type Plugins

Plugins without User Interface

- IWLDAPUserGroupPluginBusiness

  This plugin uses only the method afterUserCreateOrUpdate(User user, Group parentGroup) that is declared in the UserGroupPluginBusiness interface.

  The method afterUserCreateOrUpdate(User user, Group parentGroup) of each plugin is called after creating a new user or after adding an user to a group.

  This plugin uses the call of the method to notify a replication listener about a user change.

Plugins for Main Toolbar

The following plugins add buttons to the main toolbar of the User Application.

- CalBusiness

  Provides a personal calendar for an user to keep personal entries and appointments.

- ClubMemberExchangePlugin

  Provides methods to move a user from one group to another.

- JasperReportBusiness

  Provides access to a tool for creating reports.

- NationalRegisterBusiness

  Provides an option to import users from a file where the users are listed in a certain format. This plugin is also used for synchronizing the user information with the information from the national register.

Plugins for User Properties Pane

Plugins which add an extra tab to the user properties pane for each user.

- GroupApplicationBusiness

  Provides methods to handle applications of users for certain groups.

- StaffUserPluginBusiness

  Provides a tab for displaying and editing a user's staff information, such as title, area and education.

- UserFamilyPluginBusiness

  Provides a tab for displaying and editing a user's family information, such as spouse, children and siblings.

- UserHistoryPluginBusiness
  Provides a tab for displaying user history regarding status changes and group memberships changes.
- UserStatusPluginBusiness
  Provides a tab for displaying and editing user status.

1.6.2 Group Type Plugins

Group Type Plugins

Plugins for Group Toolbar

The following plugin adds a button to the group toolbar of the User Application.

- PinLookupToGroupImportHandler

  A simple import file handler that reads a file with personalIds and names. If the personal id number exists in the database it adds that user to the root group.

Plugins for Group Properties Pane

Plugins which add an extra tab to the group properties pane for each group.

- GroupBoardPluginBusiness

  Provides tab for managing the board members of a group.

  GroupLogoPluginBusiness

  Provides tab for managing the logo of a group. An image is set and linked to the group.

- GroupOfficeAddressPluginBusiness

  Provides tab for managing the office address of a group.

- GroupOfficeContactPluginBusiness

  Provides tab for managing office phones, email addresses, fax and homepage of a group.

Specific Plugins for Sport Union Application

- UpdateClubDivisionTemplatePluginBusiness

  This plugin adds a button to the group toolbar of the User Application within the Sport Union Application.

  Provides access to a tool for updating groups regarding to the structure of a so called template group. The structure of sport clubs (e.g., the structure of the player groups) might change due to decisions of the union. By this plugin changes do not have to be done manually for each club.

- AgeGenderPluginBusiness

  This plugin adds an extra tab to the group properties pane for each group within the Sport Union Application.

  Provides tab for displaying and handling age and gender information of a group. Sports group are often categorized regarding age and gender.

- ClubDivisionHandlerPluginBusiness

  This plugin adds an extra tab to the group properties pane for each group within the Sport Union

Application.

Provides tab for displaying and handling club division specific information such as division number and name.

- ClubInformationPluginBusiness

  This plugin adds an extra tab to the group properties pane for each group within the Sport Union Application.

  Provides tab for displaying and handling club specific information such as year when founded, regional union and if still active.

- DeildirPluginBusiness

  This plugin adds an extra tab to the group properties pane for each group within the Sport Union Application.

  Provides tab for displaying and handling club division specific information. Much more information than ClubDivisionHandlerPluginBusiness and more specific for club division than ClubInformationPluginBusiness.

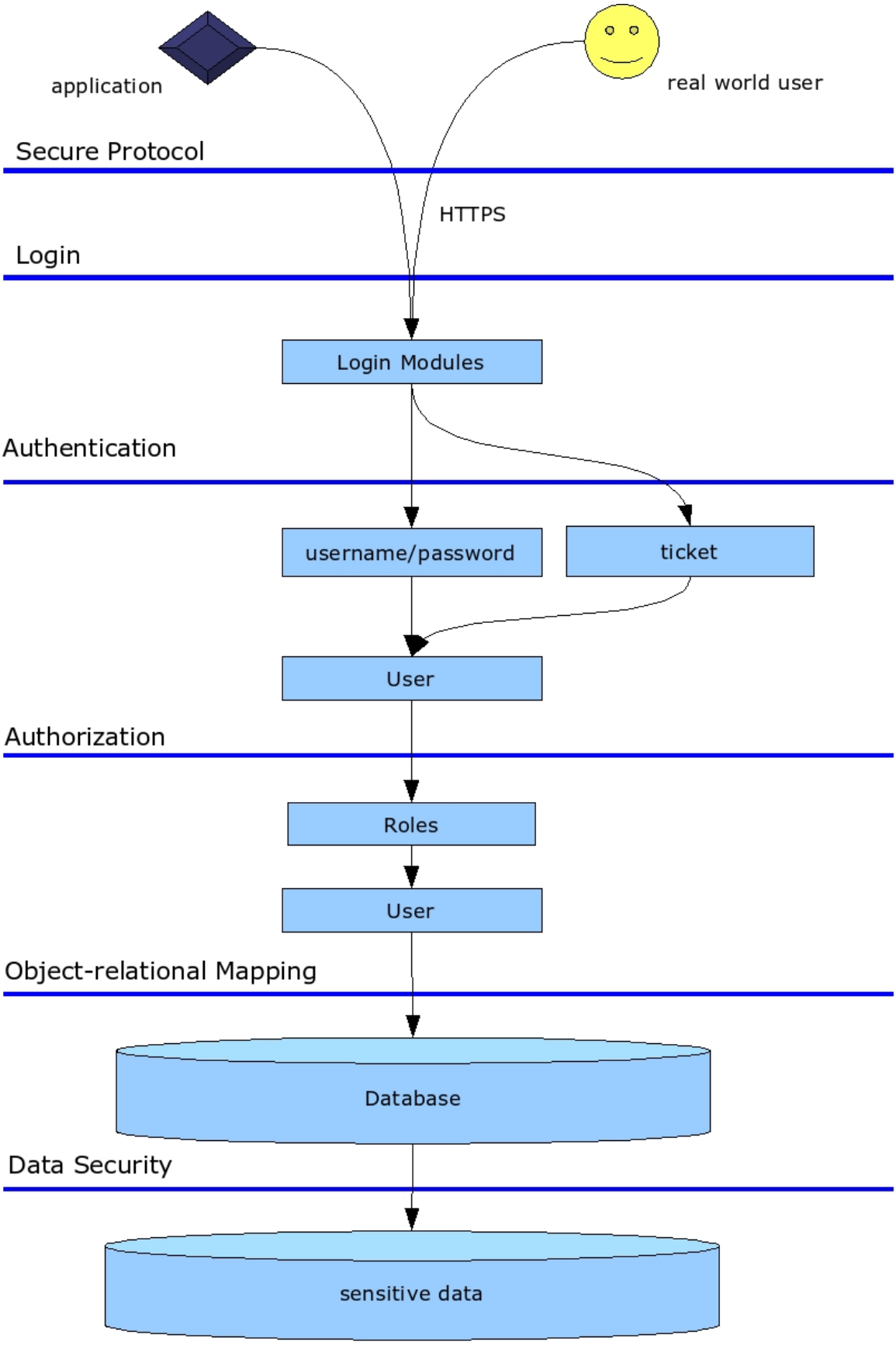
- FlokkarPluginBusiness

  This plugin adds an extra tab to the group properties pane for each group within the Sport Union Application.

  Provides tab for displaying and handling specific information for a division that contains players of the club. Such information is coach and competition type.

1.7 Security

Security



Secure Protocol

HTTPS is fully supported by idegaWeb. All modules take care of the given protocol and do not change it (e.g., by using direct links instead of relative links).

Login

Smart login modules analyzing the requests prevent attacks (e.g., the number of attempts to login is limited thus preventing brute force attacks).

Authentication

There are several methods of authentication implemented in idegaWeb, which ones are enabled depends on the kind of web application.

Basic Authentication and HTTPS

Basic authentication with HTTPS is mainly used for authenticating clients of web services. A much more sophisticated way to authenticate the client is implemented by idegaWeb in some applications by using the Web Service Security standard.

Form-based Authentication

Almost all idegaWeb applications use an approach called form-based lazy authentication that lets users navigate through unprotected areas of the site without requiring a password.

Only when the user wants to access protected areas, such as ordering or account status, the user has to login. This is done by filling out a form, that is usually only typing in a user name and password, and submitting it to the server.

The user name and password is then verified on the server side. After successful authentication a corresponding information is stored in the session. IdegaWeb uses the HttpSession object maintained by the servlet engine to keep track of already logged in users. As long as the user does not log out, the session is not destroyed or expired the user does not have to log in again.

Cookie-based Authentication

IdegaWeb has the option to offer installing a cookie on the client during login that enables the client to login the next time without submitting a form.

Note that this method is convenient for users but not recommended when user access has to be protected very well since every user using the same client is automatically logged in.

Single Sign-On

IdegaWeb supports single sign on by using a unique ticket. In that way people can log in on other systems and can simply use idegaWeb without logging in again. The ticket automatically expires after a while or can be invalidated (e.g., by logging off).

Universally Unique Identifier and Authorized Hosts

Quite similar to single sign-on a unique ticket is used to identify the user. The ticket does not expire and is used to bunch several idegaWeb applications together where the user has to login only once if the user has the same unique id accross multiple servers. A request with the ticket is only accepted when coming

from a registered server that is it is checked what ip address the request refers to.

Authorization

As already mentioned in section "Users and Groups" idegaWeb has a multiple-role-based security model. A user is assigned to one or more permission groups and then access is granted to those permission groups. IdegaWeb has system level scoped roles and module scoped ones.

Object-relational Mapping

Since all database access runs through the persistence layer using objects rather strings and direct sql statements the database itself is well protected against attacks like sql injection.

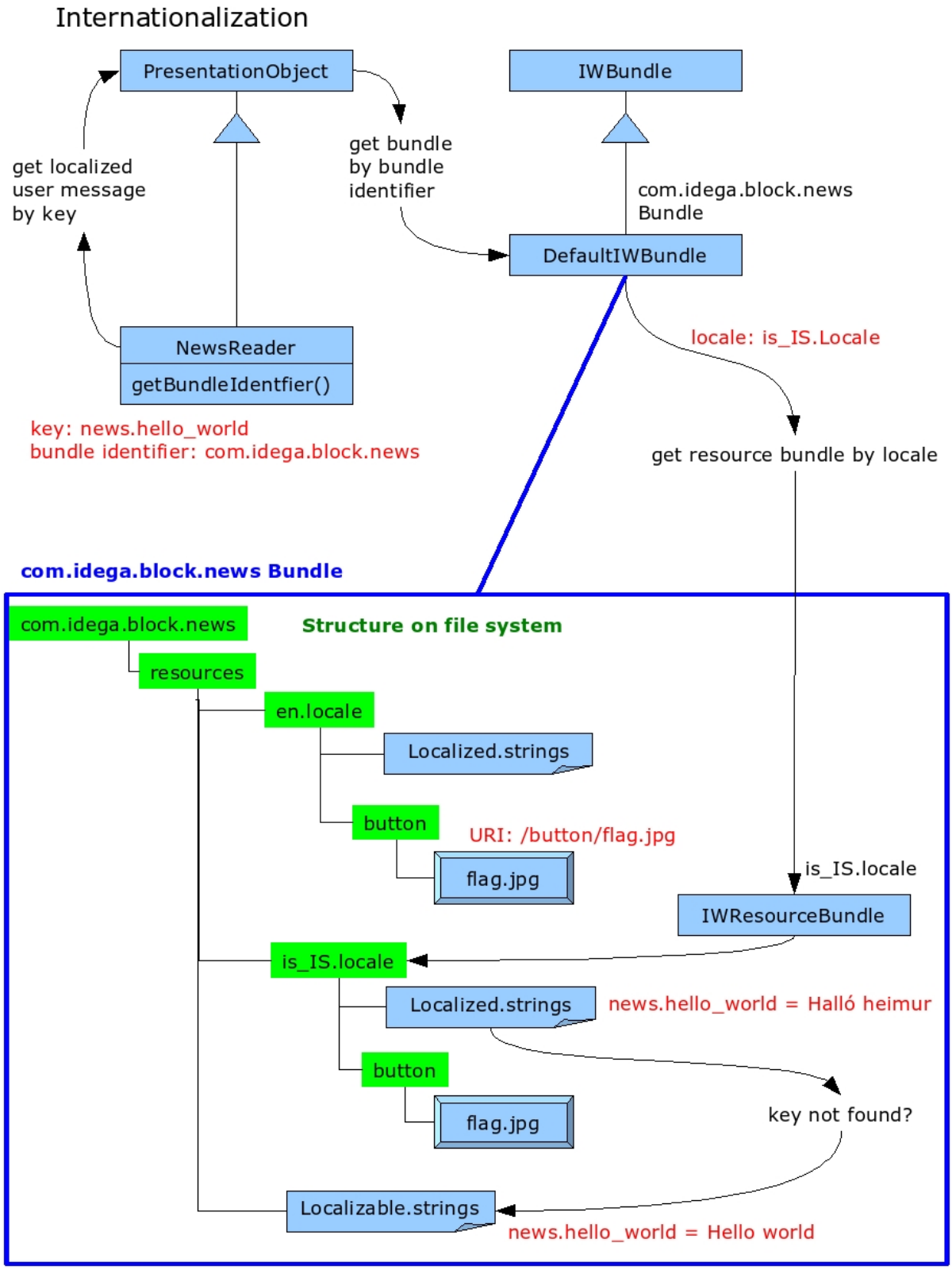
Data Security

Encrypting Passwords

All passwords in idegaWeb are stored one-way hash encrypted using the MD5 algorithm. Therefore the passwords can not be restored even if in the worst case the database table is read by an intruder.

1.8 Internationalization

Internationalization



Localized Objects

As already mentioned Java supports internationalization by using Unicode character encoding and handles local customs like different data formats.

Localized Strings

IdegaWeb uses keys for textual elements in the user interface that are replaced by localized strings during runtime. If an object needs a localized string it asks first for the corresponding bundle of the object. The bundle is found by the bundle identifier of the object.

In most cases a subclass of PresentationObject is asking for a localized string. In that case the methods of the superclass PresentationObject can be used to get the bundle. The subclass should overwrite the method getBundleIdentifier() that is called by PresentationObject and returns the bundle identifier.

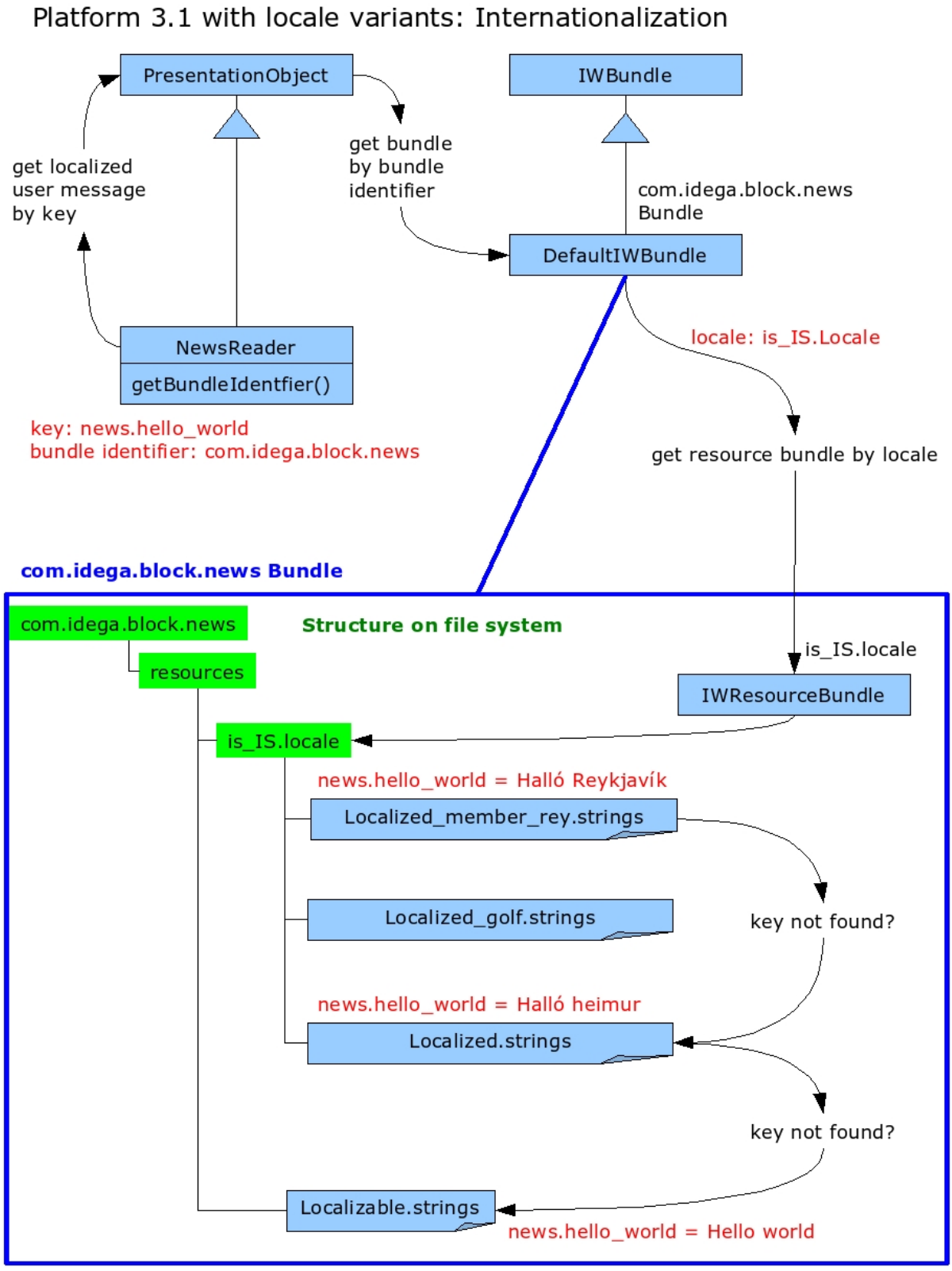
The next step is to get the current locale from the system. Depending on the locale the corresponding resource bundle is fetched from the bundle. The resource bundle is a representation of a folder on the file system that contains all resources of a bundle for that locale. The most important file among those resources is Localized.strings. The file Localized.strings contains a map of localized strings. The localized string is finally looked up in this map. If the key for a certain message could not be found in the map the special file Localizable.strings is looked up for a default string.

The special file Localizable.strings does neither belong to any specific resource bundle nor to any specific locale. It is not located in any folder of a certain locale but in the parent folder. Although the default user messages are usually written in English Localizable.strings should not be used for representing any English locale.

Other Localized Objects

For other localized objects like images the procedure for getting the right object is slightly different as described for strings. After fetching the right resource bundle the right object is retrieved by using a given URI pointing to the object. That means there is no map but a folder structure that corresponds to the URI.

Platform 3.1: Locale Variants



In platform 3.1 so called locale variants were introduced. Locale variants are used to overwrite the values for certain keys of a locale when a certain application is running.

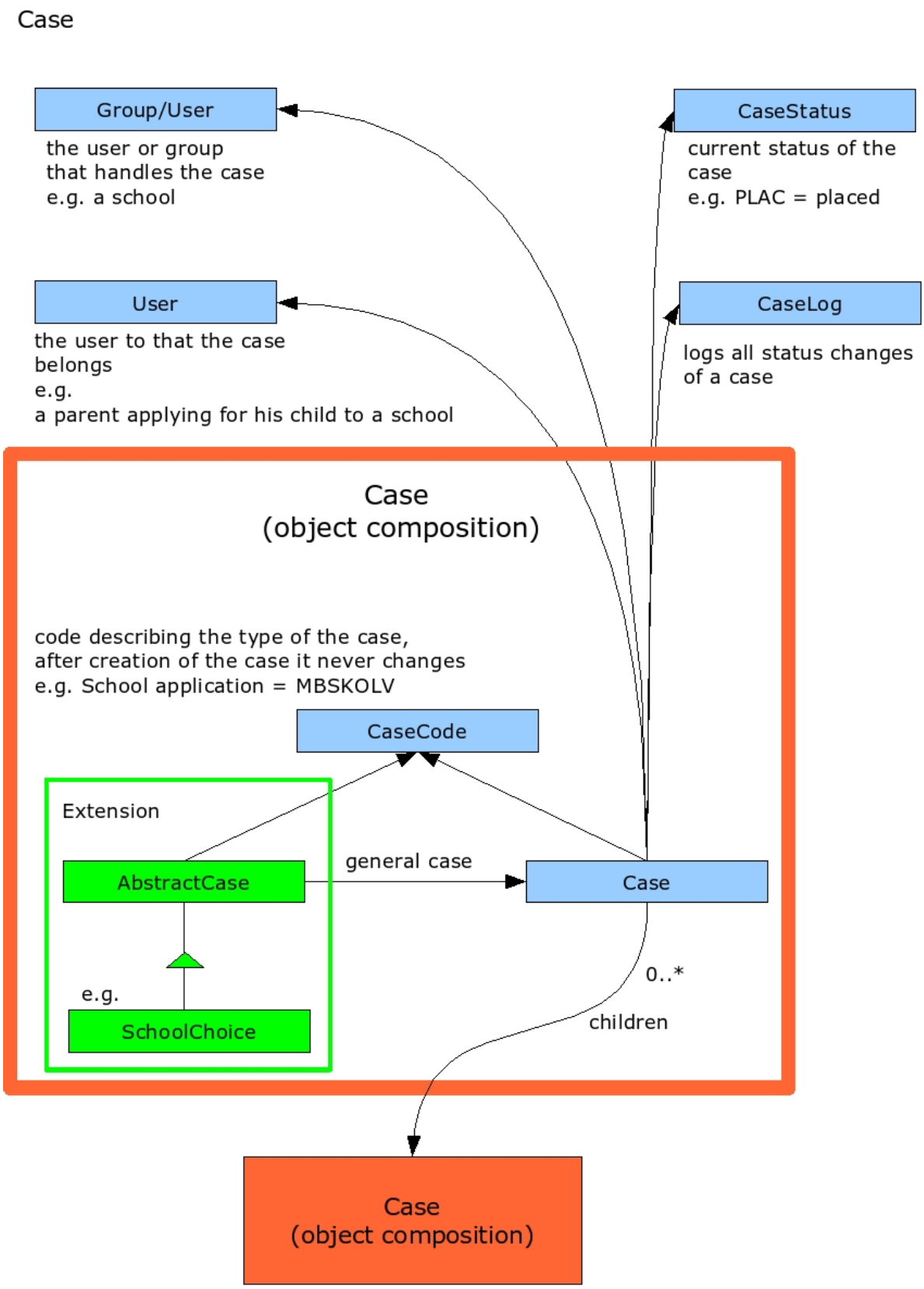
This is done by using additionally files for looking up the keys. The naming convention for those files is that the variant name follows the string "Localized_" and is then followed by the ending ".strings" (e.g., to the variant "member_rey" belongs a file named "Localized_member_rey.strings"). Those files contain like the Localized.strings files a map of localized strings. All those files are located in the corresponding locale folder. That means those files are added to the corresponding folder that contain already the Localized.strings files.

The variant is set as an application property with the key "com.idega.core.localevariant" (e.g., "com.idega.core.localevariant" is set to "member_rey").

If such an application property is set the lookup for an localized string is performed as follows. Depending on the current locale a corresponding resource bundle is fetched from the bundle as described above. Instead of looking up the file Localized.strings it is now first checked if that bundle contains a file that belongs to the specified variant. If such a file exists the key is looked up in the corresponding map. If not found the key is looked up in the corresponding map represented by the file Localized.strings in the same folder. If the key is neither found there finally the map represented by the file Localizable.strings is used for getting the value.

1.9 Case Add-on Module

Case/Process Module



IdegaWeb provides a status-based case framework. The case/process (called com.idega.block.process) is

an add-on module to the ePlatform and is intended to be used in business processes where a piece of data flows from one user to another and attributes change along the way.

This is extensively used in the eGov application modules for structuring up citizen applications.

A good example for a case is a school application. The user applies for a school and gets a response.

Use cases like a school application are represented by a composition of objects in idegaWeb. Such a composition in idegaWeb might consist of three objects.

- Case

  The general case object keeps a reference to the user to that the case belongs (e.g., the user that applied for a school).

  It has also a reference to the user or group that handles the case (e.g., the school that decides if the application is accepted).

  Every case has a state represented by the CaseStatus. The status changes during the life time of a case (e.g., starts with "Case open" and ends with "placed" or "rejected").

  Every change of a case is logged into a certain database table represented by the CaseLog object.

  Case is mapped in the PROC_CASE database table.

- CaseCode

  The CaseCode is just an object defining the type of a case. A CaseCode has a unique short string as an identifier. (e.g., a school application has a case code with the unique identifier MBSKOLV). Instead of using the object itself very often just the identifier of the corresponding CaseCode object is used.

  CaseCode is mapped in the PROC_CASE_CODE database table.

- A subclass of AbstractCase

  Each subclass of AbstractCase is written for a certain use case. It can have special attributes and special methods, thus extending the general Case object. The subclass of AbstractCase and the Case object are linked together by a reference in AbstractCase pointing to the general Case object.

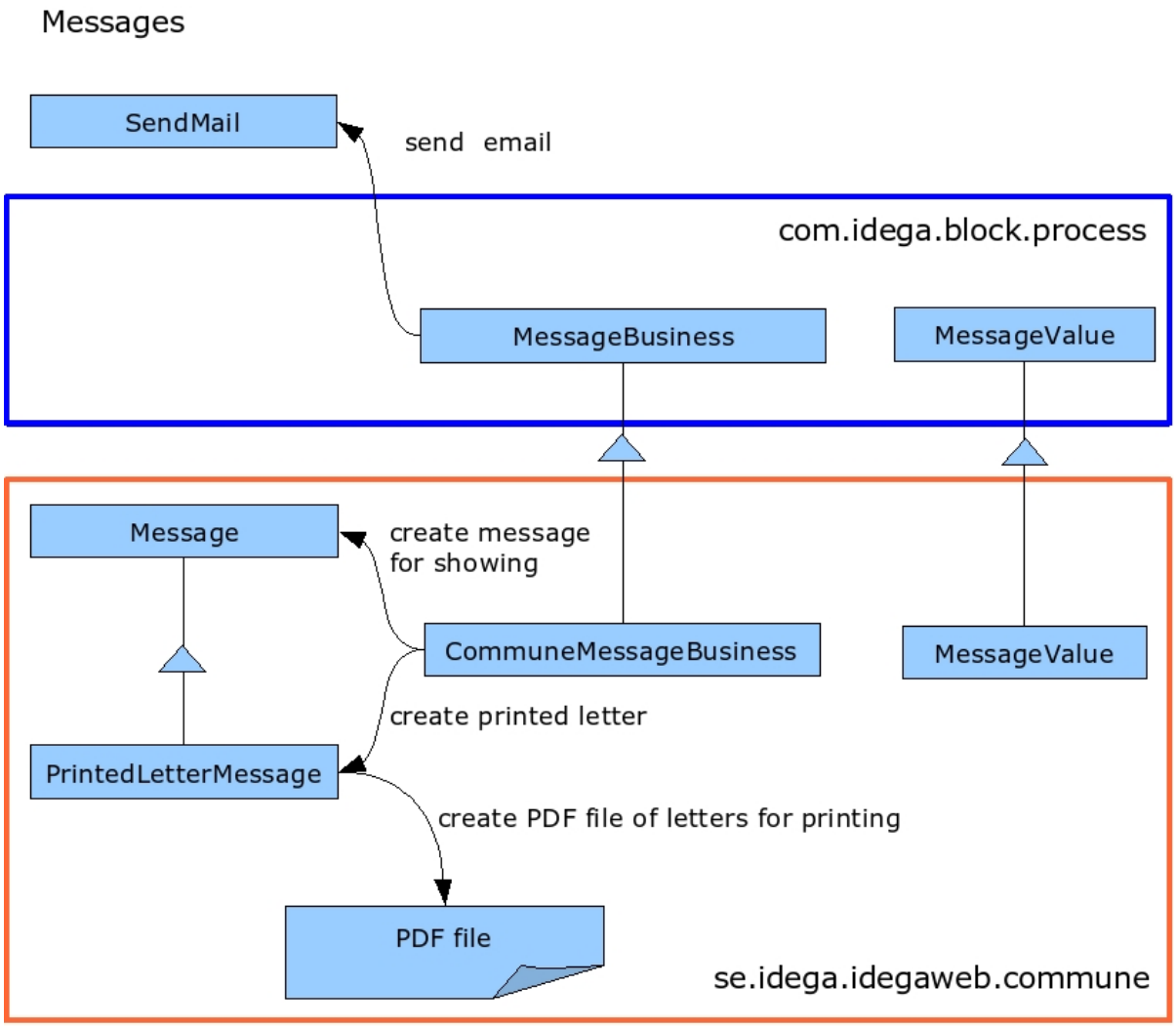
  If such a composition is used the CaseCode object seems to be redundant since the type of the use case is already defined by the used subclass of AbstractCase. However the case code is needed to map this composition to the relational database. It is clear that both the subclass of AbstractCase and the Case object are using the same CaseCode if they belong to the same composition.

  A valid use case representation in idegaWeb might be just a general Case object with the corresponding CaseCode object defining the type but in most instances the described composition is used.

Cases can have other cases as children that are added during the life of a case. Every child is again a composition as described (e.g., the school application case has usually mail messages as children. Mail messages are represented by the case object UserMessage).

1.9.1 Messages

Messages



Simple Message Handling

A simple handling of messages is done in com.idega.block.process. The most important class is MessageBusiness that creates emails with the help of the class MessageValue. The util class SendMail in com.idega.util is used for actually sending the emails.

Sophisticated Message Handling

A more sophisticated handling of messages is implemented in CommuneMessageBusiness that is extending MessageBusiness. The class MessageValue that is used by CommuneMessageBusiness is also extended but has the same name.

CommuneMessageBusiness offers three ways to handle messages:

- Sending an email

  Basically this is the same functionality like in MessageBusiness. It sends the message as email to the user by using SendMail in com.idega.util.

- Sending to the box

  Sending to the box means that the message is stored in a box that is shown when the user logs in.

- Sending a letter

  A PrintedLetterMessage is prepared, ready for sending by mail. The printing is triggered manually by the administrator of the application. If triggered by the administrator one PDF file containing all current letters is created and can be downloaded for printing.

1.10 Database

Database

Database and Object Model

When studying the database structure it is important to keep in mind that the whole database structure is derived from the object model. When installing a web application the whole database might be empty, it is created during startup depending on the entities.

Therefore looking at the object model might be much easier for understanding than looking at the database that is just the mapped representation of the object model on a relational database.

Known Issues

Problems with Oracle databases

With release 8.1.6 and higher, the Oracle JDBC drivers are compliant with the JDBC 2.0 specification.

Older Oracle JDBC drivers implementing only the JDBC 1.22 standard support LOB functionality through extensions (see package oracle.jdbc). Those extension are also provided by the current Oracle JDBC drivers that support the JDBC 2.0 standard.

IdegaWeb uses those extension to read and write LOBs into Oracle databases. To do so the returned ResultSet of a query is cast into a OracleResultSet, in that way the special extension of Oracle can be used. However when using JNDI the returned ResultSet can not be cast to a OracleResultSet.

This means that using an Oracle database with a JNDI DataSource is not supported by idegaWeb. When using Oracle the database has to be set up by using the property file db.properties.

Note that in version 3.5 of idegaWeb ePlatform this limitation will be solved. .

Database Table Names

Table names are defined in the corresponding entity class and should follow some simple rules as described below. Prefixes group the tables

All table names are starting with a prefix that says which tables belong together and what kind of data the tables contain. Tables with the same prefix form often a kind of group where most references to the tables are from tables in the same group. In most cases the corresponding entities of tables with the same prefix belong to the same bundle (e.g., IC_USER and IC_GROUP belong to the core bundle).

The following list explains the most important prefixes.

- COMM means Commune
- CON means Contract
- IB means idegaWeb Builder
- IC means idegaWeb Core
- MSG means Message

- PRO means Process
- SCH means School
- TX means Text

Table names should equal the name of the entity

Table names should just equal the name of the corresponding entity (e.g., the entity Address has the name IC_ADDRESS).

Table names should not exceed 27 letters

Since idegaWeb supports different databases table names should never exceed the limit of 27 letters to ensure that the tables can be created in every database.

The limit is set to 27 since the primary key column name (if only one is used) is by convention the table name plus the ending "_ID". If a table name is 27 letters long the corresponding name for the primary key is 30 letters long. The limit of column names is in many databases 30 letters.

Words in table names are separated by underscores

Words in the table name are separated by an underscore (e.g., SCH_SCHOOL_SEASON).

Middle tables are composed of the table names that they are referring to

Tables are created automatically by the application during startup. Table names of the entities are defined in the entity classes but the names of the middle tables are not necessarily defined in the entity classes.

Middle tables are created when there is a many-to-many relationship. A many-to-many relationship is defined in the entity class. The creation of the middle tables is done like the creation of the other tables automatically. The name of the middle table is composed by taking the names of the tables that the middle table is pointing to and linking them together putting an underscore in-between (e.g., the table SCH_SCHOOL_SCH_SCHOOL_USER represents the many-to-many relationship between the table SCH_SCHOOL and SCH_SCHOOL_USER. This relationship says that a user can be a student of more than one school. Vice versa a school has more than only one student).

If the result of such a composition exceeds 30 letters the letters at the end of the name are just cut off. Note that middle tables do not have a single primary key like the entities.

Although it is not necessary to define names for the middle tables those names should be defined in the entity classes. The same rule as described above should be applied when setting the name. If the name however exceeds the 30 letters limit another reasonable name should be used rather than cutting off letters at the end of the name.

It should be avoided to define table names for entities that could equal middle table names. Entity table names should therefore not contain other table names as part of the name unless they extend existing table names (e.g., assume there is an entity SchoolManager. The entity SchoolManager could get the table name SCH_SCHOOL_MANAGER. This name extends the existing table name SCH_SCHOOL. The name SCH_MANAGER_SCH_SCHOOL should however not be used. This name could equal a middle table name of the tables SCH_SCHOOL and SCH_MANAGER).

Column Names

Column Names should not exceed the 30 letter limit

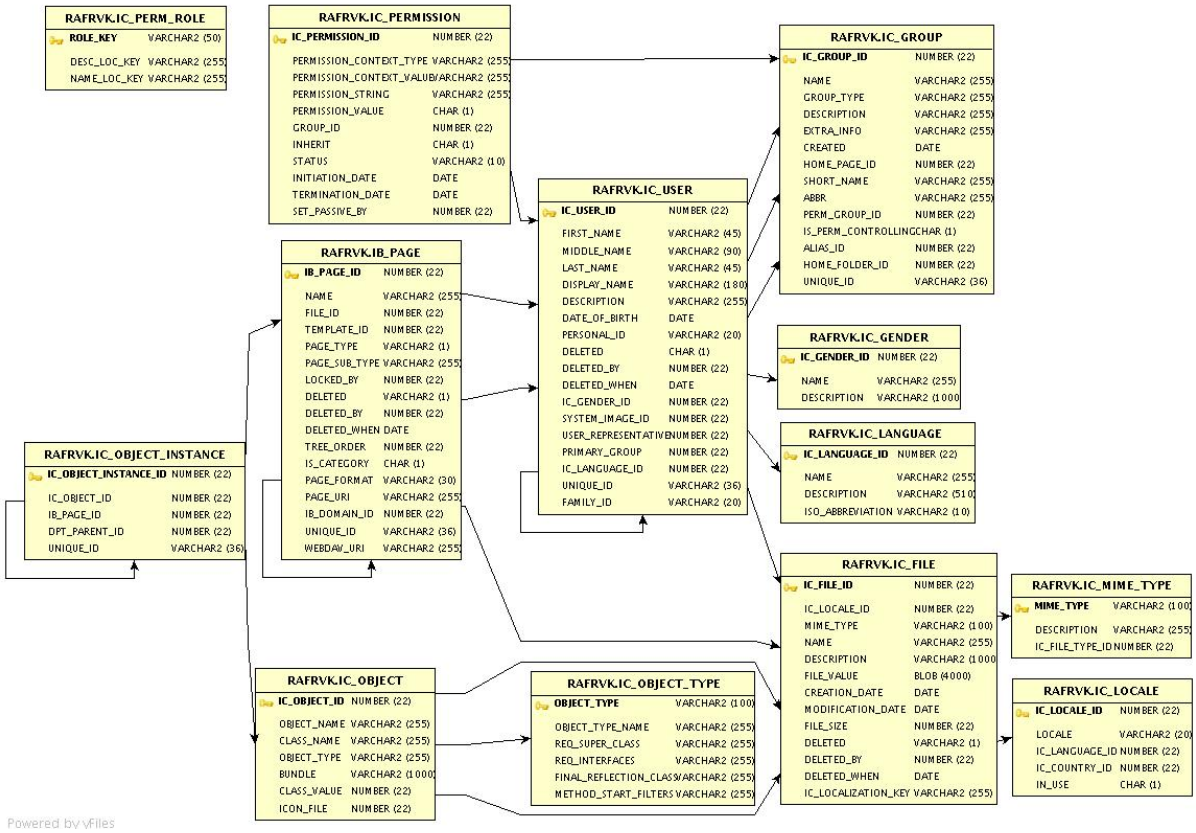
As already mentioned this limit is set to be able to support different databases.

Naming convention for the primary key

As explained above the naming convention for the primary key (if only one is used) is to use the table name and add the ending "_ID" (e.g., the table IC_USER has the primary key column name IC_USER_ID).

Important Tables

The picture below shows some of the most important tables IC_USER, IC_GROUP, IC_OBJECT, IC_OBJECT_INSTANCE, IC_FILE, IC_PERMISSION and IC_PERMISSION_ROLE.



1.11 Best Practices

Best Practices

1.11.1 Example Of Block

Example of a Block: com.idega.block.school.presentation.SchoolSeasonEditor

Super Class SchoolBlock

SchoolSeasonEditor extends the abstract SchoolBlock.

The super class SchoolBlock defines the static variable IW_BUNDLE_IDENTIFIER with the name of the bundle "com.idega.block.school". The method getBundleIdentifier() overrides the method of the super class Block and returns the bundle identifier.

SchoolBlock also overrides the main() method of the super class Block and sets some useful variables like the bundle, resource bundle and the corresponding business class for this kind of blocks that handle school stuff.

The main() method calls the abstract init() method. This method must be overridden by subclasses.

```
public abstract class SchoolBlock extends Block {

          public final static String IW_BUNDLE_IDENTIFIER =
"com.idega.block.school";

        ...some more variables...

        private IWBundle iwb;
        private IWResourceBundle iwrb;
        private SchoolBusiness business;

        public void main(IWContext iwc) throws Exception {
                this.iwb = getBundle(iwc);
                this.iwrb = getResourceBundle(iwc);
                this.business = getSchoolBusiness(iwc);
                init(iwc);
        }

        protected abstract void init(IWContext iwc) throws Exception;

        public String getBundleIdentifier(){
                return IW_BUNDLE_IDENTIFIER;
          }

          ...some more methods...
 }
```

SchoolSeasonEditor

The class SchoolSeasonEditor is an editor for editing school seasons. A list of all already defined school seasons is shown. The user can edit, delete or create seasons.

The list of seasons depends on the kind of schools that is the list depends on the category of the school.

(e.g., the seasons for music schools are not the same for sport schools).

```
public class SchoolSeasonEditor extends SchoolBlock {

        private static final String PARAMETER_ACTION = "sch_season_prm_action";
        private static final String PARAMETER_SCHOOL_SEASON_PK =
"prm_school_season_pk";
        private static final String PARAMETER_NAME = "prm_name";
        private static final String PARAMETER_CATEGORY = "prm_category";
        private static final String PARAMETER_SEASON_START = "prm_season_start";
        private static final String PARAMETER_SEASON_END = "prm_season_end";
        private static final String PARAMETER_CHOICE_START_DATE =
"prm_choice_start_date";
        private static final String PARAMETER_CHOICE_END_DATE =
"prm_choice_end_date";

        private static final int ACTION_VIEW = 1;
        private static final int ACTION_EDIT = 2;
        private static final int ACTION_NEW = 3;
        private static final int ACTION_SAVE = 4;
        private static final int ACTION_DELETE = 5;


        protected void init(IWContext iwc) throws Exception {
                switch (parseAction(iwc)) {
                        case ACTION_VIEW:
                                showList(iwc);
                                break;

                        case ACTION_EDIT:
                                Object schoolPK =
iwc.getParameter(PARAMETER_SCHOOL_SEASON_PK);
                                showEditor(iwc, schoolPK);
                                break;

                        case ACTION_NEW:
                                showEditor(iwc, null);
                                break;

                        case ACTION_SAVE:
                                saveArea(iwc);
                                showList(iwc);
                                break;

                        case ACTION_DELETE:
getBusiness().removeSchoolSeason(iwc.getParameter(PARAMETER_SCHOOL_SEASON_PK));
                                showList(iwc);
                                break;
                }
        }

        ...more methods...
}
```

The above method init() is called by the main() method of the super class SchoolBlock. The method init() calls parseAction(). The method parseAction() below parses the request. The very first time the page is requested a form was not submitted. That means the parameter PARAMETER_ACTION is not set thus this method returns ACTION_VIEW.

If the method returns ACTION_VIEW the initMethod calls showList() thus the response just shows the list of school seasons.

```
private int parseAction(IWContext iwc) {
        if (iwc.isParameterSet(PARAMETER_ACTION)) {
                return Integer.parseInt(iwc.getParameter(PARAMETER_ACTION));
        }
        return ACTION_VIEW;
}
```

If a user clicks on a season for editing the parameter PARAMETER_ACTION with the value ACTION_EDIT is send in the request. The parseAction() methods therefore returns ACTION_EDIT and the method init() calls showEditor().

The showEditor() method below composes the editor. Note the use of such simple PresentationObjects like TextInput, DateInput and SubmitButton. Note that the parameters in the constructor methods of those components are the parameters used in the request.

The call of method add() at the end of the method showEditor() belongs to the super class PresentationObjectContainer and adds the form to the children list of the SchoolSeasonEditor.

```
public void showEditor(Object seasonPK) throws RemoteException {
        Form form = new Form();
        form.setStyleClass(STYLENAME_SCHOOL_FORM);

        TextInput inputName = new TextInput(PARAMETER_NAME);
        DateInput inputStart = new DateInput(PARAMETER_SEASON_START);
        DateInput inputEnd = new DateInput(PARAMETER_SEASON_END);
        DateInput inputStartDate = new DateInput(PARAMETER_CHOICE_START_DATE);
        DateInput inputDueDate = new DateInput(PARAMETER_CHOICE_END_DATE);

        ...adding input elements to form...

        SubmitButton save =
        (SubmitButton) getButton(new SubmitButton(localize("save", "Save"),
PARAMETER_ACTION, String.valueOf(ACTION_SAVE)));
        SubmitButton cancel =
        (SubmitButton) getButton(new SubmitButton(localize("cancel", "Cancel"),
PARAMETER_ACTION, String.valueOf(ACTION_VIEW)));

        form.add(cancel);
        form.add(save);

        add(form);
}
```

The set() method below defines for what kind of schools the list is shown (e.g., for music schools). This set method is defined in the component property file so that this set method is available in the idegaWeb Builder. If this block is put on a idegaWeb Builder page by a user, the user also has to set the value of the category in the properties dialog of the idegaWeb Builder. After that the corresponding page document holds the value for this set method. In other words the category is set as a property value into the

corresponding page document.

Note that this set method is called during handling a user request when the builder page is first populated.

See "Properties of a Component - properties Folder" in chapter "Bundles".

Compare to chapter"Request Processing".

```
public void setSchoolCategory(String category) {
        this.iSchoolCategory = category;
}
```

1.11.2 Example Of Entity

Example of an Entity: se.idega.idegaweb.commune.adulteducation.data.SchoolCoursePackage

SchoolCoursePackage is an entity that represents a package of courses that a certain school is offering.

Four classes belong to SchoolCoursePackage

- interface SchoolCoursePackage
- SchoolCoursePackageBMPBean
- interface SchoolCoursePackageHome
- SchoolCoursePackageHomeImpl

Solely the class SchoolCoursePackageBMPBean is implementing all specific behaviour and specific attributes of the the entity SchoolCoursePackage. Therefore already the class SchoolCoursePackageBMPBean is sufficient to define the entity SchoolCoursePackage.

To ease development there is an idega eclipse tool that generates the bean's interface class, the bean's home interface class and the bean's home implementation class by parsing the bean class (e.g., SchoolCoursePackage, SchoolCoursePackageHome and SchoolCourseHomeImpl is generated by the idega eclipse tool just by parsing SchoolCoursePackageBMPBean. Only the class SchoolCoursePackageBMPBean is needed).

SchoolCoursePackageBMPBean

As in chapter"Persistence Components" explained SchoolCoursePackageBMPBean extends GenericEntity and implements SchoolCoursePackage.

First the columns and the entity name are defined as private static final variables. They should not be declared public since all access to the bean is done via SchoolCoursePackage and SchoolCoursePackageHome.

The entity name corresponds to the table name.

```
public class SchoolCoursePackageBMPBean extends GenericEntity implements
SchoolCoursePackage{

        private static final String ENTITY_NAME = "vux_school_package";

        private static final String COLUMN_PACKAGE = "vux_package_id";
        private static final String COLUMN_SCHOOL = "sch_school_id";
        private static final String COLUMN_SEASON = "sch_school_season_id";
        private static final String COLUMN_FREE_TEXT = "free_text";
        private static final String COLUMN_ACTIVE = "active";

        ...more methods...
}
```

The below method getEntityName() that returns the table name must be implemented.

```
/* (non-Javadoc)
 * @see com.idega.data.GenericEntity#getEntityName()
 */
public String getEntityName() {
        return ENTITY_NAME;
}
```

The below method initializeAttributes() must be implemented.

```
/* (non-Javadoc)
 * @see com.idega.data.GenericEntity#initializeAttributes()
 */
public void initializeAttributes() {
        addAttribute(getIDColumnName());

        addManyToOneRelationship(COLUMN_PACKAGE, CoursePackage.class);
        addManyToOneRelationship(COLUMN_SCHOOL, School.class);
        addManyToOneRelationship(COLUMN_SEASON, SchoolSeason.class);

        addAttribute(COLUMN_FREE_TEXT, "Free text", String.class);}
        addAttribute(COLUMN_ACTIVE, "Active", Boolean.class);

        addManyToManyRelationShip(AdultEducationCourse.class);
}
```

The method initializeAttribute() is called during startup of the web application to create or update the database table.

The inherited method getIDColumnName() should be used to create the primary key column. It adds "_ID" to the entity name to compose the column name for the primary key. For this class the primary key column is therefore "VUX_SCHOOL_PACKAGE_ID". The table name is "VUX_SCHOOL_PACKAGE".

The methods addAttribute() define the attributes of the entity. Each attribute corresponds to a column.

The method addManyToOneRelationship() adds a many-to-one relationship to the entity (e.g., addManyToOneRelationship(COLUMN_PACKAGE, CoursePackage.class) adds a many-to-one relationship to SchoolCoursePackage. This relationsship says that a SchoolCoursePackage points to one CoursePackage but a CoursePackage can point to more than one SchoolCoursePackage. This means that the same course package might be offered by several schools).

The method addManyToManyRelationShip() adds a many-to-many relationship to AdultEducationCourse. It says that a package contains more than one course and that the same course might be in different packages.

The method addManyToManyRelationShip() creates a middle table in the database to represent the many-to-many relationship.

If the database is empty a call of the method initAttributes() creates the following tables in the database

- table VUX_SCHOOL_PACKAGE

  This table represents the entity SchoolCoursePackage.

  It has the following columns

  - VUX_SCHOOL_PACKAGE_ID, primary key
  - VUX_PACKAGE_ID, points to SchoolPackage, many-to-one relationship. It is good style to choose the same name as the primary key column of the target entity
  - SCH_SCHOOL_ID, points to a School, many-to-one relationship
  - SCH_SCHOOL_SEASON_ID, points to a season, many-to-one relationship
  - FREE_TEXT, free text
  - ACTIVE, a boolean, character "Y" or "N" is stored into the database

- table VUX_COURSE_VUX_SCHOOL_PACKAGE

  This is the middle table that represents the many-to-many relationship between SchoolCoursePackage and AdultEducationCourse. AdultEducationCourse has the entity name "vux_course".

  The middle table has the following both columns

  - VUX_COURSE_ID, points to an AdultEducationCourse, same name as the primary key column of that entity
  - VUX_SCHOOL_PACKAGE_ID, points to a SchoolCoursePackage, same name as the primary key column of that entity

The below method setDefaultValues() overrides a method in GenericEntity that does nothing. Here it sets the value of the attribute active to false. That means the course package is defined but not yet available for students. In general this method is used to set default values of an entity.

```
public void setDefaultValues() {
        setActive(false);
}
```

Below are the get methods of the bean listed. Note the different return types.

```
//Getters
public CoursePackage getPackage() {
        return (CoursePackage) getColumnValue(COLUMN_PACKAGE);
}

public Object getPackagePK() {
        return getIntegerColumnValue(COLUMN_PACKAGE);
}

public School getSchool() {
        return (School) getColumnValue(COLUMN_SCHOOL);
}
```

```
public Object getSchoolPK() {
        return getIntegerColumnValue(COLUMN_SCHOOL);
}

public SchoolSeason getSeason() {
        return (SchoolSeason) getColumnValue(COLUMN_SEASON);
}

public Object getSeasonPK() {
        return getIntegerColumnValue(COLUMN_SEASON);
}

public String getFreeText() {
        return getStringColumnValue(COLUMN_FREE_TEXT);
}

public boolean isActive() {
        return getBooleanColumnValue(COLUMN_ACTIVE, true);
}

public Collection getCourses() throws IDORelationshipException {
        return idoGetRelatedEntities(AdultEducationCourse.class);
}
```

Below are the set methods of the bean listed. Note the different parameter types.

```
//Setters
public void setPackage(CoursePackage coursePackage) {
        setColumn(COLUMN_PACKAGE, coursePackage);
}

public void setSchool(School school) {
        setColumn(COLUMN_SCHOOL, school);
}

public void setSeason(SchoolSeason season) {
        setColumn(COLUMN_SEASON, season);
}

public void setFreeText(String text) {
        setColumn(COLUMN_FREE_TEXT, text);
}

public void setActive(boolean active) {
        setColumn(COLUMN_ACTIVE, active);
}
```

Below the add and remove methods for the many-to-many relationship between SchoolCoursePackage and AdultEducationCourse are listed.

The parameter collection of the method addCourses() must be a collection of AdultEducationCourses.

```
public void addCourse(AdultEducationCourse course) throws
IDOAddRelationshipException {
```

```
        idoAddTo(course);
}

public void addCourses(Collection courses) throws IDOAddRelationshipException {
        Iterator iter = courses.iterator();
        while (iter.hasNext()) {
                addCourse((AdultEducationCourse) iter.next());
        }
}

public void removeCourse(AdultEducationCourse course) throws
IDORemoveRelationshipException {
        idoRemoveFrom(course);
}

public void removeCourses() throws IDORemoveRelationshipException {
        idoRemoveFrom(AdultEducationCourse.class);
}
```

Below are special find methods of the bean listed. Those methods are called indirectly by using the corresponding home interface. See paragraph below.

```
//Finders
public Collection ejbFindAllBySchool(School school) throws FinderException {
        Table table = new Table(this);

        SelectQuery query = new SelectQuery(table);
        query.addColumn(table, getIDColumnName());
        query.addCriteria(new MatchCriteria(table, COLUMN_SCHOOL,
MatchCriteria.EQUALS, school));

        return idoFindPKsByQuery(query);
}

public Collection ejbFindBySchoolAndSeason(School school, SchoolSeason season)
throws FinderException {
        Table table = new Table(this);

        SelectQuery query = new SelectQuery(table);
        query.addColumn(table, getIDColumnName());
        query.addCriteria(new MatchCriteria(table, COLUMN_SCHOOL,
MatchCriteria.EQUALS, school));
        query.addCriteria(new MatchCriteria(table, COLUMN_SEASON,
MatchCriteria.EQUALS, season));

        return idoFindPKsByQuery(query);
}

public Collection ejbFindBySchoolAndSeasonAndPackage(School school, SchoolSeason
season, CoursePackage coursePackage) throws FinderException {
        Table table = new Table(this);

        SelectQuery query = new SelectQuery(table);
        query.addColumn(table, getIDColumnName());
        query.addCriteria(new MatchCriteria(table, COLUMN_SCHOOL,
MatchCriteria.EQUALS, school));
        if (season != null) {
                query.addCriteria(new MatchCriteria(table, COLUMN_SEASON,
```

```
MatchCriteria.EQUALS, season));
        }
        if (coursePackage != null) {
                query.addCriteria(new MatchCriteria(table, COLUMN_PACKAGE,
MatchCriteria.EQUALS, coursePackage));
        }

        return idoFindPKsByQuery(query);
}

public int ejbHomeGetNumberOfSchoolPackages(CoursePackage coursePackage) throws
IDOException {
        Table table = new Table(this);

        SelectQuery query = new SelectQuery(table);
        query.addColumn(new CountColumn(table, getIDColumnName()));
        query.addCriteria(new MatchCriteria(table, COLUMN_PACKAGE,
MatchCriteria.EQUALS, coursePackage));

        return idoGetNumberOfRecords(query);
}
```

SchoolCoursePackageHomeImpl

Below are the methods listed that every home implementation must implement. Note that for such three methods

- getting the interface class
- creating an entity
- finding an entity by an identifier

no methods of SchoolCoursePackageBMPBean are called but only methods of the super class IDOFactory are involved.

```
public class SchoolCoursePackageHomeImpl extends IDOFactory implements
SchoolCoursePackageHome {

        protected Class getEntityInterfaceClass() {
                return SchoolCoursePackage.class;
        }

        public SchoolCoursePackage create() throws javax.ejb.CreateException {
                return (SchoolCoursePackage) super.createIDO();
        }

        public SchoolCoursePackage findByPrimaryKey(Object pk) throws
javax.ejb.FinderException {
                return (SchoolCoursePackage) super.findByPrimaryKeyIDO(pk);
        }

        ...more special methods...

}
```

Below are the special methods of SchoolCoursepackage listed. Note the call of the methods with prefix "ejb". This is the place where the above mentioned find methods are involved.

Note the very special method for getting just the number of course packages.

```
public Collection findAllBySchool(School school) throws FinderException {
        com.idega.data.IDOEntity entity = this.idoCheckOutPooledEntity();
        java.util.Collection ids = ((SchoolCoursePackageBMPBean)
entity).ejbFindAllBySchool(school);
        this.idoCheckInPooledEntity(entity);
        return this.getEntityCollectionForPrimaryKeys(ids);
}

public Collection findBySchoolAndSeason(School school, SchoolSeason season) throws
FinderException {
        com.idega.data.IDOEntity entity = this.idoCheckOutPooledEntity();
        java.util.Collection ids = ((SchoolCoursePackageBMPBean)
entity).ejbFindBySchoolAndSeason(school, season);
        this.idoCheckInPooledEntity(entity);
        return this.getEntityCollectionForPrimaryKeys(ids);
}

public Collection findBySchoolAndSeasonAndPackage(School school, SchoolSeason
season, CoursePackage coursePackage)
                throws FinderException {
        com.idega.data.IDOEntity entity = this.idoCheckOutPooledEntity();
        java.util.Collection ids = ((SchoolCoursePackageBMPBean)
entity).ejbFindBySchoolAndSeasonAndPackage(school, season,
                        coursePackage);
        this.idoCheckInPooledEntity(entity);
        return this.getEntityCollectionForPrimaryKeys(ids);
}

public int getNumberOfSchoolPackages(CoursePackage coursePackage) throws
IDOException {
        com.idega.data.IDOEntity entity = this.idoCheckOutPooledEntity();
        int theReturn = ((SchoolCoursePackageBMPBean)
entity).ejbHomeGetNumberOfSchoolPackages(coursePackage);
        this.idoCheckInPooledEntity(entity);
        return theReturn;
}
```

1.12 Developer Tools

Developer Tools

This chapter lists some tools commonly used by developers.

1.12.1 Platform 2: Developer Tools

Platform 2: Developer Tools

The following lists describe developer tools available under platform 2.

Most Used And Important Developer Tools

- Localizer

  Localizer is a tool for creating or changing localized strings per bundle.

- Locale Setter

  Locale Setter sets which languages should be available on the server. The above mentioned Locale Switcher offers only the available languages. Locale Setter sets also the default language that is the language the server should use when responding the first request of a user.

- Application Property Setter

  Tool for managing application properties. Beside handling predefined application properties new properties can be set.

  Since behaviour of many web applications is defined by some applications properties this tool is quite important and useful.

- Bundle Components / Bundle Components Manager

  Tool for managing components of bundles.

  If a new component is added to a bundle an entry is put into bundle.pxml and the corresponding component property file is created (e.g., if the component "com.idega.block.text.presentation.TextReader" is added to a bundle a "com.idega.block.text.presentation.TextReader.pxml" component property file is created).

  By this tool a class of a bundle can be registered as a component of a bundle. By registering the component it becomes available in the idegaWeb Builder and can be put on a web page.

  See also paragraph "Properties of a Component - properties Folder" in chapter "Bundles".

- Component Manager

  Tool for managing the properties of a component of a bundle. (e.g., a component of the type "iw.element" has properties specifying available settings in the idegaWeb Builder for the corresponding component).

  For example new set methods of a component can be registered by this tool. After registering they are available in the idegaWeb Builder.

  See also paragraph "Properties of a Component - properties Folder" in chapter "Bundles".

- SQL Querer

  Rich database tool to examine the underlying database. Note that this tool does not support more

than one database yet.

Other Developer Tools

- Locale Switcher

  Locale Switcher just changes the used language for the current session. Locale Switcher is actually a module that can be put on a web page in the idegaWeb builder. In that way every user can choose its preferred language.

- Object Types / Object Type Manager

  Object Types /Object Type Manager sets the available object types within the application. This tool refers directly to entries in the table ic_object_type. Usually entries of this table are added and maintained automatically by the application. Administrators should carefully change or add entries if at all. Object types are for example iw.element or iw.block.

- Bundle Creator

  Used for creating and registering new bundles. Following folders and files are created when submitting a bundle identifier (e.g., "com.idega.block.text") for the new bundle.

  - /*bundle identifier*.bundle

    The folder with the name of the bundle. The suffix ".bundle" is added automatically.

  - the folder /*bundle identifier*.bundle/classes
  - the folder /*bundle identifier*.bundle/properties
  - the folder /*bundle identifier*.bundle/resources
  - the file /*bundle identifier*.bundle/bundle.pxml

    The bundle property file bundle.pxml contains only the bundle identifier at the beginning.

- Bundle Property Setter

  Properties of a bundle can be set with this tool. Those properties are stored into the bundle.pxml file in the corresponding bundle folder.

- Bundle Resource Manager

  Tool for managing the content of the resource folder of a bundle. Files can be deleted and uploaded. Folders can be deleted and created. The view of the tool depends on a chosen locale. If non locale is chosen resources that belong to all locales are shown. If a certain locale is chosen only the resources that belong to the chosen locale are shown. (e.g., if locale "English" is chosen only the content of the folder en.locale is shown. The folder en.locale is not shown if non locale is chosen or if another locale is chosen).

- DBPool Status Viewer

  Viewer that shows the status of the database pool.

- Application Status

  Shows following values of the application

  - last startup

- last shutdown
- last restart
- uptime
- maximal uptime

Note that the restart button is experimental.

- Caches
  See special paragraph "Caches".
- Logs
  Tool for looking into the error log and out log files. Note that response time might be very long.
- Versions
  Experimental tool for showing versions of bundles.
- Update Manager
  Simple Update Manager for updating modules. This manager works without support of any versions of the bundles.
- Homepage Generator
  Special tool that is used only for a certain project. Can not be used for any other projects.
- Page Object Viewer
  Tool for examining idegaWeb Builder pages. Searches for components used on idegaWeb Builder pages.
- Script Manager
  This tool manages a list of available Beanshell scripts (end with .bsh) within bundles and includes a simple script editor and the possibility to run the scripts.
- LDAP Manager
  Tool for handling all settings of the LDAP functionality.

1.12.2 Update Manager

Platform3: Update Manager

Update Manager is a tool for administrators to update or add new modules to a web application.

Update Manager is only available in version 3 and above. It is accessible from under the Manager tab in the idegaWeb Workspace.

The Update Manager has the following steps.

- Connecting to a repository with username and password.

  In the first step the administrator is asked for a repository URL, username and password. The Update Manager is then connecting to the repository using basic authentication.

- Updating modules or installing new modules

  In the second step the administrator is asked if he would like to update existing modules or install new ones. So there are two options.

  - Option "Updating"

    - Choosing modules

      In the third step a list is created which versions of modules are available for updating. The administrator chooses the desired versions of different modules that he wants to update.

  - Option "Installing new modules"

    - Choosing modules

      In the third step a list is created which new modules are available. Version of the modules are not shown yet. The administrator chooses the desired modules that he wants to install.

    - Choosing version

      In the forth step a list is created which versions of the previous chosen modules are available for installing. The administrator chooses the desired versions of different modules that he wants to install.

- Confirming installation

  In the last step a list is created which versions of other modules are needed to be installed because of dependencies. Also the modules that were chosen are shown. The administrator has to confirm the installation.

- Performing installation

  After confirming the installation the chosen modules plus the needed ones are installed and the web application is restarted after around 20 seconds.

1.12.3 Caches

Caches

The Caches menu item is accessible from within the "Developer".

Clearing caches is usually done only for debugging, during development or special testing.

IdegaWeb Builder Pages

Caching of idegaWeb Builder pages is explained in chapter "Requesting an idegaWeb Builder page".

It is harmless to clear the cache when running an application.

Lookup Cache

Lookups of entity beans and service beans are cached in the Lockup Cache.

Since certain entries are put into the cache by bundle starters during startup clearing the cache is not harmless and an application won't work proper afterwards.

The option for clearing the cache was removed in platform 3.1.

Bean Cache

Bean caching can be set in application settings as active or non-active.

Since there are unsolved problems regarding handling LOBs bean caching is set to be non-active on production servers. Nevertheless some special beans use this cache due to performance reasons even if bean caching is set to non-active.

It is not clear if there might be some side effects when clearing the cache.

This cache should not be active on production servers so that no beans are using it apart from the above mentioned special beans.

Query Cache

Caching queries can be set in application settings as active or non-active.

Since there are unsolved problems like handling joins query caching is set to be non-active on production servers.

This cache should not be active on production servers.

IWCacheManager Cache

IWCacheManager caches blocks as described in subparagraph "Caching of Blocks" in chapter "UI Components".

Due to some unsolved problems regarding notifying blocks it is not safe to clear the cache.

IWCacheManager2

IWCacheManager2 is based on Ehcache and is implemented in platform 3.1. It replaces the old cache for builder pages, lookup cache, bean cache and query cache. Therefore clearing that cache means clearing all the mentionend ones.

1.13 Installation

Installation

Much information of this chapter was extracted from "Apache Tomcat Application Developer's Guide". Thus this chapter mainly refers to installing on Tomcat.

Webarchive File

A web application is defined as a hierarchy of directories and files in a standard layout. Such a hierarchy can be accessed in its "unpacked" form, where each directory and file exists in the file system separately, or in a "packed" form known as a Web ARchive, or WAR file. The former format is more useful during development, while the latter is used when the application is distributed to be installed.

The top-level directory of the web application hierarchy is called the document root (*DOCUMENT_ROOT*) of the application.

Directory Layout Of an idegaWeb Application

The directory layout of idegaWeb is organized as follows

- *DOCUMENT_ROOT*/idegaweb/bundles

  The place where the root folders of the used bundles (e.g., "/com.idega.block.forum") are located.

- *DOCUMENT_ROOT*/style

  The place where general style files are located. This folder is only used in older web applications and is now being considered as deprecated.

  Files regarding style (e.g., css files) are put into the style folder under the resources folder in the corresponding bundle.

- *DOCUMENT_ROOT*/WEB_INF/web.xml

  The web.xml file is the web application deployment descriptor of the web application. This XML file describes the servlets and other components that make up the application, along with any initialization parameters and container-managed security constraints the server is enforced.

  This file is a composition of many web.xml files that are defined in several bundles.

- *DOCUMENT_ROOT*/WEB-INF/classes/

  Contains only the log4j.properties property file.

- *DOCUMENT_ROOT*/WEB-INF/lib/

  This directory contains the jar files of the bundles and all other jar files that are needed.

- *DOCUMENT_ROOT*/WEB-INF/idegaweb/properties/

  This directory contains the following files

  - db.properties

Files that holds the piece of information how to connect to a database. See also chapter "Persistence Layer".

product.properties

Property file with properties regarding the web application product (e.g., build id, version)

- idegaweb.pxml

  Property file with general application settings.

  See also paragraph below.

- system_properties.pxml

  Property file with properties regarding general application settings.

  See also paragraph below.

- *DOCUMENT_ROOT*/WEB-INF/idegaweb/properties/users

  This directory contains the properties files of the users.

  Note that user properties are not stored in the database. That means that they have to be temporarily saved when updating the web application by installing a new war file, because the web application folder (DOCUMENT_ROOT) is overwritten during installation.

  See also paragraph "User Properties" in chapter "User and Groups".

Property File idegaweb.pxml

The file idegaweb.pxml is a property file with general application settings.

The main focus of this property file is on technical aspects (e.g., debug mode, caching mode).

The API class to read property values from this file is called IWMainApplicationSettings. Note that this class also tries to read and write those properties from the table IC_APPLICATION_BINDING.

Property File system_properties.pxml

The file system_properties.pxml is a property file with properties regarding general application settings.

Main focus is on aspects of the application domain (e.g., main start date). This property file is rarely used since properties regarding the domain are mostly set in those bundle property files of bundles where they are actually being used.

The reason this exists beyond the standard idegaweb.pxml is that this file and API allows for a hierarchical structure in properties and categorizing and has a standard UI Component for modifying those called SystemProperties.

Note that originally this file was only read and written from within the web application folder. This can be problematic when updating the web application by installing a new war file, because the web application folder (DOCUMENT_ROOT) is overwritten during installation. So this behaviour has been modified so the file is originally at first installation read from the web application folder. After that it is

persisted in the database in the IC_FILE table. Henceforth the stored property file in the database is used. IWSystemProperties class implements the described behaviour.

Creating a Webarchive File For Distribution>

When creating a webarchive file the described directory layout should be used.

Web Application Deployment Descriptor - web.xml file

The web.xml file has to be created by composing all web.xml files that are defined in the bundles.

See also paragraph "Web Application Deployment Descriptor - WEB-INF/web.xml File" in chapter "Bundles".

Database

The db.properties file should not be added to the war file to be able to use the same war file for different web servers. Instead either JNDI for looking up the database should be used or the db.properties file should be added later when the web server has extracted the war flle the very first time. That means that the web application has to be started twice, first without a connection to the database and then again with a connection to the database.

Installation Using Apache Tomcat

In order to be executed, a web application like idegaWeb must be deployed on a servlet container. This is true even during development. Here it is described using Tomcat 5 to provide the execution environment. A web application can be deployed in Tomcat as follows.

The description below uses the name *CATALINA_HOME* to refer to the directory into which Tomcat 5 was installed, and is the base directory against which most relative paths are resolved. However, if Tomcat 5 is configured for multiple instances by setting a *CATALINA_BASE* directory, *CATALINA_BASE* should be used instead of *CATALINA_HOME* for each of these references.

- During development

  The unpacked directory hierarchy is copied into a subdirectory in directory *CATALINA_HOME*/webapps/. Tomcat will assign a context path to the application based on the subdirectory name that is chosen. Tomcat has to be restarted after installing or updating the application.

- During distribution

  The web application archive file (war file) is copied into the directory *CATALINA_HOME*webapps/. When Tomcat is started, it will automatically expand the web application archive file into its unpacked form, and execute the application that way.

  If this approach is used and the application should be updated later, both the web application archive file has to be replaced and the expanded directory that Tomcat created has to be deleted. After that Tomcat has to be restarted in order to reflect the changes.

Installation Using Other Web Containers

Deploying a web application on other servlet containers will be specific to each container, but all containers compatible with the Servlet API Specification (version 2.3 or later) are required to accept a

web application archive file. Note that other containers are not required to accept an unpacked directory structure (as Tomcat does), but this feature is commonly available.

1.14 Examples Of web.xml

Examples Of web.xml

1.14.1 Platform 2.0: web.xml

Platform 2.0: web.xml

```
    <?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
        PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

        <description>IdegaWeb ePlatform v.2</description>
        <display-name>IdegaWeb-ePlatform-v.2</display-name>

<!-- MODULE:BEGIN com.idega.core 1.0 -->


  <!-- idegaWeb event listener that starts the application -->
  <listener>
      <listener-class>com.idega.idegaweb.IWApplicationStarter</listener-class>
  </listener>


  <servlet>
    <servlet-name>Excel</servlet-name>
    <display-name>Excel</display-name>
    <servlet-class>com.idega.io.ExcelOutput</servlet-class>
  </servlet>

  <servlet>
    <servlet-name>ObjectInstanciator</servlet-name>
    <display-name>ObjectInstanciator</display-name>
    <servlet-class>com.idega.servlet.ObjectInstanciator</servlet-class>
    <load-on-startup>4</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>PageInstanciator</servlet-name>
    <display-name>PageInstanciator</display-name>
    <servlet-class>com.idega.servlet.PageInstanciator</servlet-class>
    <load-on-startup>3</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>WindowOpener</servlet-name>
    <display-name>WindowOpener</display-name>
    <servlet-class>com.idega.servlet.WindowOpener</servlet-class>
    <load-on-startup>6</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>IWEventHandler</servlet-name>
    <display-name>IWEventHandler</display-name>
    <servlet-class>com.idega.servlet.IWEventHandler</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>IWEventHandler</servlet-name>
    <url-pattern>/servlet/IWEventHandler</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>Excel</servlet-name>
```

```
  <url-pattern>/excel</url-pattern>
</servlet-mapping>


<servlet-mapping>
  <servlet-name>ObjectInstanciator</servlet-name>
  <url-pattern>/servlet/ObjectInstanciator</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ObjectInstanciator</servlet-name>
  <url-pattern>/servlet/ObjectInstanciator/</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ObjectInstanciator</servlet-name>
  <url-pattern>/servlet/ObjectInstanciator/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>PageInstanciator</servlet-name>
  <url-pattern>/servlet/PageInstanciator</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>WindowOpener</servlet-name>
  <url-pattern>/servlet/WindowOpener</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>WindowOpener</servlet-name>
  <url-pattern>/servlet/WindowOpener/</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>WindowOpener</servlet-name>
  <url-pattern>/servlet/WindowOpener/*</url-pattern>
</servlet-mapping>

  <!-- The IWEncodingFilter Filter , MUST be the first filter -->
  <filter>
      <filter-name>IWEncodingFilter</filter-name>
      <filter-class>com.idega.servlet.filter.IWEncodingFilter</filter-class>
  </filter>

  <filter-mapping>
      <filter-name>IWEncodingFilter</filter-name>
      <url-pattern>/*</url-pattern>
  </filter-mapping>



  <!-- DISABLED The GZIPFilter that compressed output to the client
  <filter>
      <filter-name>compress</filter-name>
      <filter-class>com.idega.servlet.filter.GZIPFilter</filter-class>
  </filter>

  <filter-mapping>
      <filter-name>compress</filter-name>
      <url-pattern>*.jsp</url-pattern>
  </filter-mapping>

  <filter-mapping>
      <filter-name>compress</filter-name>
      <url-pattern>*.html</url-pattern>
```

```
    </filter-mapping>

        <filter-mapping>
        <filter-name>compress</filter-name>
        <url-pattern>*.htm</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>compress</filter-name>
        <url-pattern>/pages/*</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>compress</filter-name>
        <url-pattern>/servlet/*</url-pattern>
    </filter-mapping>

    <filter-mapping>
    <filter-name>compress</filter-name>
    <url-pattern>/workspace/*</url-pattern>
    </filter-mapping>

    -->
   <!-- End of GZIPFilter -->

        <welcome-file-list>
                <welcome-file>index.jsp</welcome-file>
                <welcome-file>index.html</welcome-file>
        </welcome-file-list>

        <!-- Definition of the standard Default DataSource -->
        <resource-ref>
                <description>Default DB Connection</description>
                <res-ref-name>jdbc/DefaultDS</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
        </resource-ref>

<!-- MODULE:END com.idega.core 1.0 -->

<!-- MODULE:BEGIN org.apache.axis 1.0 -->


<!-- A slightly modified version of the web.xml that comes with Axis. Modified by
Eirikur S. Hrafnsson, eiki@idega.is -->
    <listener>
<listener-class>org.apache.axis.transport.http.AxisHTTPSessionListener</listener-class>
    </listener>

  <servlet>
    <servlet-name>AxisServlet</servlet-name>
    <display-name>Apache-Axis Servlet</display-name>
    <servlet-class>
        org.apache.axis.transport.http.AxisServlet
    </servlet-class>
  </servlet>

  <servlet>
    <servlet-name>AdminServlet</servlet-name>
    <display-name>Axis Admin Servlet</display-name>
    <servlet-class>
        org.apache.axis.transport.http.AdminServlet
```

```
    </servlet-class>
    <load-on-startup>100</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>SOAPMonitorService</servlet-name>
    <display-name>SOAPMonitorService</display-name>
    <servlet-class>
        org.apache.axis.monitor.SOAPMonitorService
    </servlet-class>
    <init-param>
      <param-name>SOAPMonitorPort</param-name>
      <param-value>5001</param-value>
    </init-param>
    <load-on-startup>100</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>/servlet/AxisServlet</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>*.jws</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>SOAPMonitorService</servlet-name>
    <url-pattern>/SOAPMonitor</url-pattern>
  </servlet-mapping>

<!-- uncomment this if you want the admin servlet -->
<!--
  <servlet-mapping>
    <servlet-name>AdminServlet</servlet-name>
    <url-pattern>/servlet/AdminServlet</url-pattern>
  </servlet-mapping>
-->

<!-- Commented out because this is defined in the webapp
    <session-config>
         <session-timeout>5</session-timeout>
    </session-config>
-->

    <!-- currently the W3C havent settled on a media type for WSDL;
    http://www.w3.org/TR/2003/WD-wsdl12-20030303/#ietf-draft
    for now we go with the basic 'it's XML' response -->
  <mime-mapping>
    <extension>wsdl</extension>
     <mime-type>text/xml</mime-type>
  </mime-mapping>


  <mime-mapping>
    <extension>xsd</extension>
```

```
    <mime-type>text/xml</mime-type>
  </mime-mapping>

<!-- MODULE:END org.apache.axis 1.0 -->


<!-- MODULE:BEGIN com.idega.block.media 1.0 -->


  <servlet>
    <servlet-name>MediaServlet</servlet-name>
    <display-name>MediaServlet</display-name>
    <servlet-class>com.idega.block.media.servlet.MediaServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>MediaServlet</servlet-name>
    <url-pattern>/servlet/MediaServlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>MediaServlet</servlet-name>
    <url-pattern>/servlet/MediaServlet/</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>MediaServlet</servlet-name>
    <url-pattern>/servlet/MediaServlet/*</url-pattern>
  </servlet-mapping>

<!-- MODULE:END com.idega.block.media 1.0 -->

<!-- MODULE:BEGIN com.idega.builder 1.0 -->


  <servlet>
    <servlet-name>IBMainServlet</servlet-name>
    <display-name>IBMainServlet</display-name>
    <servlet-class>com.idega.builder.servlet.IBMainServlet</servlet-class>
    <load-on-startup>-2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>IBIFrameServlet</servlet-name>
    <display-name>IBIFrameServlet</display-name>
    <servlet-class>com.idega.builder.servlet.IBIFrameServlet</servlet-class>
    <load-on-startup>-1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/builder</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/builder/</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/builder/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/IBMainServlet</url-pattern>
```

```
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/IBMainServlet/</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/IBMainServlet/*</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>IBIFrameServlet</servlet-name>
    <url-pattern>/servlet/IBIFrameServlet</url-pattern>
  </servlet-mapping>

<!-- MODULE:END com.idega.builder 1.0 -->

</web-app>
```

1.14.2 Platform 3.1: web.xml

Platform 3.1: web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
        PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

        <description>ePlatform</description>
        <display-name>ePlatform</display-name>

<!-- MODULE:BEGIN org.apache.myfaces 1.0 -->


    <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>client</param-value>
        <description>
            State saving method: "client" or "server" (= default)
            See JSF Specification 2.5.2
        </description>
    </context-param>

    <context-param>
        <param-name>org.apache.myfaces.ALLOW_JAVASCRIPT</param-name>
        <param-value>true</param-value>
        <description>
            This parameter tells MyFaces if javascript code should be allowed in the
            rendered HTML output.
            If javascript is allowed, command_link anchors will have javascript code
            that submits the corresponding form.
            If javascript is not allowed, the state saving info and nested
parameters
            will be added as url parameters.
            Default: "true"
        </description>
    </context-param>

    <context-param>
        <param-name>org.apache.myfaces.DETECT_JAVASCRIPT</param-name>
        <param-value>false</param-value>
        <description>
            This parameter tells MyFaces if javascript code should be allowed in the
            rendered HTML output.
            If javascript is allowed, command_link anchors will have javascript code
            that submits the corresponding form.
            If javascript is not allowed, the state saving info and nested
parameters
            will be added as url parameters.
            Default: "false"

            Setting this param to true should be combined with STATE_SAVING_METHOD
"server" for
            best results.

            This is an EXPERIMENTAL feature. You also have to enable the detector
```

```
filter/filter mapping below to get
            JavaScript detection working.
        </description>
    </context-param>

    <context-param>
        <param-name>org.apache.myfaces.PRETTY_HTML</param-name>
        <param-value>true</param-value>
        <description>
            If true, rendered HTML code will be formatted, so that it is "human
readable".
            i.e. additional line separators and whitespace will be written, that do
not
            influence the HTML code.
            Default: "true"
        </description>
    </context-param>

    <context-param>
        <param-name>org.apache.myfaces.AUTO_SCROLL</param-name>
        <param-value>false</param-value>
        <description>
            If true, a javascript function will be rendered that is able to restore
the
            former vertical scroll on every request. Convenient feature if you have
pages
            with long lists and you do not want the browser page to always jump to
the top
            if you trigger a link or button action that stays on the same page.
            Default: "false"
        </description>
    </context-param>

    <!-- WelcomeFile Filter -->
    <!--
    <filter>
        <filter-name>WelcomeFile Filter</filter-name>
<filter-class>org.apache.myfaces.webapp.filter.WelcomeFileFilter</filter-class>
        <description>
            Due to the manner in which the JSP / servlet lifecycle
            functions, it is not currently possible to specify default
            welcome files for a web application and map them to the
            MyFacesServlet.  Normally they will be mapped to the
            default servlet for the JSP container.  To offset this
            shortcoming, we utilize a servlet Filter which examines
            the URI of all incoming requests.
        </description>
    </filter>
    -->

    <!-- JavaScriptDetector Filter -->
<!--    <filter>-->
<!--        <filter-name>javascriptDetector</filter-name>-->
<!--
<filter-class>org.apache.myfaces.webapp.filter.JavaScriptDetectorFilter</filter-class>-->
<!--    </filter>-->



    <!-- MyFaces Extensions Filter MOVED TO CORE -->

    <!-- Filter Mappings -->
```

```
    <!-- see MyFaces Filter above for a description -->
    <!--
    <filter-mapping>
        <filter-name>WelcomeFile Filter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    -->

    <!-- Listener, that does all the startup work (configuration, init). -->
    <listener>
<listener-class>org.apache.myfaces.webapp.StartupServletContextListener</listener-class>
    </listener>

    <!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>


    <!-- Faces Servlet Mapping -->

    <!-- virtual path mapping -->
    <!--
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    -->

    <!-- extension mapping -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>

    <!-- Welcome files - Commented out for now
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    -->
<!-- MODULE:END org.apache.myfaces 1.0 -->


<!-- MODULE:BEGIN com.idega.core 1.0 -->


    <!-- idegaWeb event listener that starts the application -->
    <listener>
        <listener-class>com.idega.idegaweb.IWApplicationStarter</listener-class>
    </listener>


  <servlet>
    <servlet-name>Excel</servlet-name>
    <display-name>Excel</display-name>
    <servlet-class>com.idega.io.ExcelOutput</servlet-class>
  </servlet>
```

```
<servlet>
  <servlet-name>ObjectInstanciator</servlet-name>
  <display-name>ObjectInstanciator</display-name>
  <servlet-class>com.idega.servlet.ObjectInstanciator</servlet-class>
  <load-on-startup>4</load-on-startup>
</servlet>
<servlet>
  <servlet-name>PageInstanciator</servlet-name>
  <display-name>PageInstanciator</display-name>
  <servlet-class>com.idega.servlet.PageInstanciator</servlet-class>
  <load-on-startup>3</load-on-startup>
</servlet>
<servlet>
  <servlet-name>WindowOpener</servlet-name>
  <display-name>WindowOpener</display-name>
  <servlet-class>com.idega.servlet.WindowOpener</servlet-class>
  <load-on-startup>6</load-on-startup>
</servlet>
<servlet>
  <servlet-name>IWEventHandler</servlet-name>
  <display-name>IWEventHandler</display-name>
  <servlet-class>com.idega.servlet.IWEventHandler</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>IWEventHandler</servlet-name>
  <url-pattern>/servlet/IWEventHandler</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>Excel</servlet-name>
  <url-pattern>/excel</url-pattern>
</servlet-mapping>


<servlet-mapping>
  <servlet-name>ObjectInstanciator</servlet-name>
  <url-pattern>/servlet/ObjectInstanciator</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ObjectInstanciator</servlet-name>
  <url-pattern>/servlet/ObjectInstanciator/</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>ObjectInstanciator</servlet-name>
  <url-pattern>/servlet/ObjectInstanciator/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>PageInstanciator</servlet-name>
  <url-pattern>/servlet/PageInstanciator</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>WindowOpener</servlet-name>
  <url-pattern>/servlet/WindowOpener</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>WindowOpener</servlet-name>
  <url-pattern>/servlet/WindowOpener/</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>WindowOpener</servlet-name>
```

```
    <url-pattern>/servlet/WindowOpener/*</url-pattern>
  </servlet-mapping>

    <!-- The IWEncodingFilter Filter , MUST be the first filter -->
    <filter>
        <filter-name>IWEncodingFilter</filter-name>
        <filter-class>com.idega.servlet.filter.IWEncodingFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>IWEncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- IWBlockFilter Filter blocks certain sensitive URLs -->
    <filter>
        <filter-name>IWBlockFilter</filter-name>
        <filter-class>com.idega.servlet.filter.IWBlockFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>IWBlockFilter</filter-name>
        <url-pattern>*.properties</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>IWBlockFilter</filter-name>
        <url-pattern>*.pxml</url-pattern>
    </filter-mapping>

    <!-- The IWUrlRedirector Filter -->
    <filter>
        <filter-name>IWUrlRedirector</filter-name>
        <filter-class>com.idega.servlet.filter.IWUrlRedirector</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>IWUrlRedirector</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- The IWAuthenticator Filter -->
    <filter>
        <filter-name>IWAuthenticator</filter-name>
        <filter-class>com.idega.servlet.filter.IWAuthenticator</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>IWAuthenticator</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- The IWBundleResourceFilter Filter -->
    <filter>
        <filter-name>IWBundleResourceFilter</filter-name>
        <filter-class>com.idega.servlet.filter.IWBundleResourceFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>IWBundleResourceFilter</filter-name>
        <url-pattern>/idegaweb/*</url-pattern>
    </filter-mapping>

    <!-- The IWWelcomeFilter Filter -->
```

```
    <filter>
        <filter-name>IWWelcomeFilter</filter-name>
        <filter-class>com.idega.servlet.filter.IWWelcomeFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>IWWelcomeFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- The IWAuthorizationFilter Filter -->
    <filter>
        <filter-name>IWAuthorizationFilter</filter-name>
        <filter-class>com.idega.servlet.filter.IWAuthorizationFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>IWAuthorizationFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

     <!-- MyFaces Extensions Filter -->
    <filter>
        <filter-name>extensionsFilter</filter-name>
<filter-class>org.apache.myfaces.component.html.util.ExtensionsFilter</filter-class>
        <init-param>
            <param-name>uploadMaxFileSize</param-name>
            <param-value>100m</param-value>
            <description>Set the size limit for uploaded files.
                Format: 10 - 10 bytes
                        10k - 10 KB
                        10m - 10 MB
                        1g - 1 GB
            </description>
        </init-param>
        <init-param>
            <param-name>uploadThresholdSize</param-name>
            <param-value>100k</param-value>
            <description>Set the threshold size - files
                    below this limit are stored in memory, files above
                    this limit are stored on disk.

                Format: 10 - 10 bytes
                        10k - 10 KB
                        10m - 10 MB
                        1g - 1 GB
            </description>
        </init-param>
<!--         <init-param>
            <param-name>uploadRepositoryPath</param-name>
            <param-value>/temp</param-value>
            <description>Set the path where the intermediary files will be stored.
            </description>
        </init-param>-->
    </filter>

    <filter-mapping>
        <filter-name>extensionsFilter</filter-name>
        <url-pattern>/pages/*</url-pattern>
    </filter-mapping>
        <filter-mapping>
                <filter-name>extensionsFilter</filter-name>
```

```
                <url-pattern>/workspace/*</url-pattern>
        </filter-mapping>
        <filter-mapping>
                <filter-name>extensionsFilter</filter-name>
                <url-pattern>/window/*</url-pattern>
        </filter-mapping>
        <filter-mapping>
                <filter-name>extensionsFilter</filter-name>
                <url-pattern>/setup/*</url-pattern>
        </filter-mapping>
        <filter-mapping>
                <filter-name>extensionsFilter</filter-name>
                <url-pattern>/faces/*</url-pattern>
        </filter-mapping>
        <filter-mapping>
                <filter-name>extensionsFilter</filter-name>
                <url-pattern>*.jsp</url-pattern>
        </filter-mapping>

    <!-- MyFaces MultiPart Filter MUST BE AFTER THE IWEncodingFilter -->

    <!-- The IWActionHandlerFilter that redirects "action" urls like
"/idegaweb/action/edit/files/cms/article/1.xml"-->
    <filter>
        <filter-name>iwActionURI</filter-name>
<filter-class>com.idega.servlet.filter.IWActionURIHandlerFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>iwActionURI</filter-name>
        <url-pattern>/idegaweb/action/*</url-pattern>
    </filter-mapping>



    <!-- DISABLED The GZIPFilter that compressed output to the client
    <filter>
        <filter-name>compress</filter-name>
        <filter-class>com.idega.servlet.filter.GZIPFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>compress</filter-name>
        <url-pattern>*.jsp</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>compress</filter-name>
        <url-pattern>*.html</url-pattern>
    </filter-mapping>

        <filter-mapping>
        <filter-name>compress</filter-name>
        <url-pattern>*.htm</url-pattern>
    </filter-mapping>

    <filter-mapping>
        <filter-name>compress</filter-name>
        <url-pattern>/pages/*</url-pattern>
    </filter-mapping>

    <filter-mapping>
```

```
        <filter-name>compress</filter-name>
        <url-pattern>/servlet/*</url-pattern>
    </filter-mapping>

    <filter-mapping>
    <filter-name>compress</filter-name>
    <url-pattern>/workspace/*</url-pattern>
    </filter-mapping>

    -->
   <!-- End of GZIPFilter -->

        <welcome-file-list>
                <welcome-file>index.jsp</welcome-file>
                <welcome-file>index.html</welcome-file>
        </welcome-file-list>

        <!-- Definition of the standard Default DataSource -->
        <resource-ref>
                <description>Default DB Connection</description>
                <res-ref-name>jdbc/DefaultDS</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
        </resource-ref>

<!-- MODULE:END com.idega.core 1.0 -->


<!-- MODULE:BEGIN com.idega.faces 1.0 -->


    <!-- idegaWeb URLs: -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/pages/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/window/*</url-pattern>
    </servlet-mapping>

        <!-- Fix to make this work with JSF -->
        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>/login/*</url-pattern>
        </servlet-mapping>

        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>/login</url-pattern>
        </servlet-mapping>

        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>/workspace/*</url-pattern>
        </servlet-mapping>

        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>/workspace</url-pattern>
        </servlet-mapping>
```

```
    <!--Listener for initializing the idegaWeb JSF extensions: -->
    <listener>
        <listener-class>com.idega.faces.IWFacesInstaller</listener-class>
    </listener>

<!-- MODULE:END com.idega.faces 1.0 -->


<!-- MODULE:BEGIN org.apache.axis 1.0 -->


<!-- A slightly modified version of the web.xml that comes with Axis. Modified by
Eirikur S. Hrafnsson, eiki@idega.is -->
    <listener>
<listener-class>org.apache.axis.transport.http.AxisHTTPSessionListener</listener-class>
    </listener>

  <servlet>
    <servlet-name>AxisServlet</servlet-name>
    <display-name>Apache-Axis Servlet</display-name>
    <servlet-class>
        org.apache.axis.transport.http.AxisServlet
    </servlet-class>
  </servlet>

  <servlet>
    <servlet-name>AdminServlet</servlet-name>
    <display-name>Axis Admin Servlet</display-name>
    <servlet-class>
        org.apache.axis.transport.http.AdminServlet
    </servlet-class>
    <load-on-startup>100</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>SOAPMonitorService</servlet-name>
    <display-name>SOAPMonitorService</display-name>
    <servlet-class>
        org.apache.axis.monitor.SOAPMonitorService
    </servlet-class>
    <init-param>
      <param-name>SOAPMonitorPort</param-name>
      <param-value>5001</param-value>
    </init-param>
    <load-on-startup>100</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>/servlet/AxisServlet</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>*.jws</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
  </servlet-mapping>
```

```
  <servlet-mapping>
    <servlet-name>SOAPMonitorService</servlet-name>
    <url-pattern>/SOAPMonitor</url-pattern>
  </servlet-mapping>

 <!-- uncomment this if you want the admin servlet -->
 <!--
  <servlet-mapping>
    <servlet-name>AdminServlet</servlet-name>
    <url-pattern>/servlet/AdminServlet</url-pattern>
  </servlet-mapping>
 -->

 <!-- Commented out because this is defined in the webapp
    <session-config>
         <session-timeout>5</session-timeout>
    </session-config>
 -->

    <!-- currently the W3C havent settled on a media type for WSDL;
    http://www.w3.org/TR/2003/WD-wsdl12-20030303/#ietf-draft
    for now we go with the basic 'it's XML' response -->
  <mime-mapping>
    <extension>wsdl</extension>
     <mime-type>text/xml</mime-type>
  </mime-mapping>


  <mime-mapping>
    <extension>xsd</extension>
    <mime-type>text/xml</mime-type>
  </mime-mapping>

<!-- MODULE:END org.apache.axis 1.0 -->


<!-- MODULE:BEGIN com.idega.graphics 1.0 -->


          <filter>
                  <filter-name>SVGFilter</filter-name>
                  <filter-class>com.idega.graphics.filter.SVGFilter</filter-class>
          </filter>

    <!-- Disabled for now: should be avoided as some browsers accept viewing svg
natively
          <filter-mapping>
                    <filter-name>SVGFilter</filter-name>
                  <url-pattern>*.svg</url-pattern>
          </filter-mapping>
          -->
    <!-- This is the default type for rendering out to PNG -->
          <filter-mapping>
                    <filter-name>SVGFilter</filter-name>
                  <url-pattern>*.psvg</url-pattern>
          </filter-mapping>
          <filter-mapping>
                    <filter-name>SVGFilter</filter-name>
                  <url-pattern>*.svg.jsp</url-pattern>
          </filter-mapping>
          <filter-mapping>
                    <filter-name>SVGFilter</filter-name>
```

```
                <url-pattern>*.svg.jspx</url-pattern>
          </filter-mapping>

        <!-- special mappings for server side generated jsp svg files:-->
            <servlet-mapping>
                <servlet-name>jsp</servlet-name>
                <url-pattern>*.jsvg</url-pattern>
            </servlet-mapping>
          <filter-mapping>
                    <filter-name>SVGFilter</filter-name>
                  <url-pattern>*.jsvg</url-pattern>
          </filter-mapping>

<!-- MODULE:END com.idega.graphics 1.0 -->
<!-- MODULE:BEGIN org.apache.slide 1.0 -->

    <!-- Definition and configuration of servlet filters -->
    <!-- Changes made by idega have comment that contain 'iw:'-->
    <filter>
        <filter-name>webdavlog</filter-name>
        <filter-class>org.apache.slide.webdav.filter.LogFilter</filter-class>
        <init-param>
            <param-name>logFormat</param-name>
            <param-value>%T, %t, %P, %m, %s "%l", %i, %p</param-value>
            <description>
                Defines the format of a log line.
                The following placeholders are available:
                    %T=thread-name,
                    %t=date-time,
                    %P=principal-name,
                    %m=method-name,
                    %s=status-code,
                    %l=default-status-text,
                    %L=detailed-status-text,
                    %i=elapsed-time,
                    %p=relative-request-uri,
                    %u=request-uri.
                    %x=request-content-length.
                    %A=header User-Agent.
            </description>
        </init-param>
        <init-param>
            <param-name>outputToConsole</param-name>
            <param-value>false</param-value>
            <description>If true, output is directed to STDOUT.</description>
        </init-param>
        <init-param>
            <param-name>outputToServletLog</param-name>
            <param-value>false</param-value>
            <description>If true, output is directed to the servlet's log
file.</description>
        </init-param>
        <!--
        <init-param>
            <param-name>outputToFile</param-name>
            <param-value>c:\webdav.log.xml</param-value>
            <description>If present, output is directed to the specified
file.</description>
        </init-param>
        -->
    </filter>
    <!-- If you're operating Slide with an SSL connection and with authentication
```

```
enabled
         and you notice that Internet Explorer is unable to open some file types you
may
         want to uncomment the following filter and its associated filter-mapping.
See the
         javadoc for the NoCacheFilter class for a description of the problem and
discussion
         of the ramifications. -->
    <!--
    <filter>
        <filter-name>nocache</filter-name>
        <filter-class>org.apache.slide.webdav.filter.NoCacheFilter</filter-class>
    </filter>
    -->

    <filter-mapping>
        <filter-name>webdavlog</filter-name>
        <servlet-name>webdav</servlet-name>
    </filter-mapping>
    <!--
    <filter-mapping>
        <filter-name>nocache</filter-name>
        <servlet-name>webdav</servlet-name>
    </filter-mapping>
    -->
    <!-- Definition and configuration of Slide's WebDAV servlet mapped to scope
/files -->
    <servlet>
        <servlet-name>webdav</servlet-name>
        <display-name>Slide DAV Server</display-name>
        <servlet-class>org.apache.slide.webdav.WebdavServlet</servlet-class>
        <init-param>
            <param-name>domain</param-name>
<param-value>/idegaweb/bundles/org.apache.slide.bundle/properties/Domain.xml</param-value><!--
iw: changed from Domain.xml to 'bundlepath'/properties/Domain.xml -->
            <description>
                Path to the domain configuration file, relative to the path of the
                web application.
                The default is '/Domain.xml'.
            </description>
        </init-param>
        <init-param>
            <param-name>namespace</param-name>
            <param-value>slide</param-value>
            <description>
                Name of the Slide namespace that should be accessed by this servlet.
                If this parameter is provided, make sure the corresponding namespace
                is defined in the domain configuration file. Otherwise, the default
                namespace will be used, if one exists.
            </description>
        </init-param>
        <init-param>
            <param-name>scope</param-name>
            <param-value/>
            <description>
                 Scope of the Slide namespace that should be exposed by this
servlet.
                 For example, if you want to expose only the /files collection via
                 WebDAV, set this parameter to '/files'. In that case, any URLs of
the
                 form '/context-path/servlet-path/*' will be mapped to '/files/*'
                 in the Slide namespace.
```

```
                The default value is an empty string.
            </description>
        </init-param>
        <init-param>
            <param-name>depth-limit</param-name>
            <!-- iw: changed from 3 to 1000 (a lot closer to infinity than 3 ;) -->
            <param-value>1000</param-value>
            <description>
                This init-parameter determines the depth limit for PROPFIND and
other
                methods, to avoid performance hits on the server for requests with
                infinite depth.
                The default value is '3'.
            </description>
        </init-param>
        <init-param>
            <param-name>default-mime-type</param-name>
            <param-value>application/octet-stream</param-value>
            <description>
                The MIME type that should be used for resources of unknown type. For
                example, if a WebDAV client uploads a file (via PUT) without
specifying
                the Content-Type header, the MIME type defined here will be used.
                The default value is 'application/octet-stream'.
            </description>
        </init-param>
        <init-param>
            <param-name>default-servlet</param-name>
            <param-value>false</param-value><!-- iw: changed from true to false -->
            <description>
                By default, the WebDAV servlet is mapped as default servlet of the
                web application context (the url-pattern in servlet-mapping is '/').
                If you want to change that mapping so the servlet is no longer the
                default servlet, you must change this initialization parameter to
                indicate the situation to the servlet, by setting it to 'false'.
                The default value is 'true'.
            </description>
        </init-param>
        <init-param>
            <param-name>directory-browsing</param-name>
            <param-value>false</param-value>
            <description>
                Use the 'directory-browsing' init-parameter to turn off generation
of
                HTML index pages that enable browsing of collections (by setting
this
                parameter to 'false'), or to specify a web-app relative path to a
                template resource (a JSP page, for example) which should handle
                generation of the HTML index page. In the latter case, you can use
a
                JSP page at WEB-INF/index.jsp by specifying '/WEB-INF/index.jsp' as
                value of this parameter.
                The default value is 'true'.
            </description>
        </init-param>
        <init-param>
            <param-name>directory-browsing-hide-acl</param-name>
            <param-value>true</param-value>
            <description>
                Use this parameter to hide ACL information in generated HTML index
pages.
                (see parameter "directory-browsing")
```

```
                The default value is 'true'.
            </description>
        </init-param>
        <init-param>
            <param-name>directory-browsing-hide-locks</param-name>
            <param-value>true</param-value>
            <description>
                Use this parameter to hide locking information in generated HTML
index pages.
                (see parameter "directory-browsing")
                The default value is 'true'.
            </description>
        </init-param>
        <init-param>
            <param-name>optimizePropfindOutput</param-name>
            <param-value>true</param-value>
            <description>
                If set to false, the PropFindMethod will first create a (large) JDOM
                document in memory and then write it to the response stream.
                If set true, the PropFindMethod will write results to the stream as
                soon as they are available. This will reduce memory consumption
                in the case of large responses (PROPFIND on many resources).
                The output of these two variants differ slightly, since in optimized
                mode the D:DAV namespace is declared in the multistatus element AND
                in all response elements. Since this is still a valid XML document
                it shouldn't be a problem, but in case you encounter any diffculties
                this switch provides a way to get around it.
            </description>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
        </init-param>
        <init-param>
            <param-name>extendedAllprop</param-name>
            <param-value>false</param-value>
            <description>
            According to RFC3253 (DeltaV), RFCxxxx (ACL) and RFCxxxx (Binding),
            a DAV:allprop PROPFIND should not return any of the properties defined
            in any of that documents.
            For testing purposes, the specified behaviour can be disabled by
            setting this parameter "true".
            </description>
        </init-param>
        <init-param>
            <param-name>lockdiscoveryIncludesPrincipalURL</param-name>
            <param-value>false</param-value>
            <description>
                As proposed on February 08, 2003 by Lisa Dusseault in
                w3c-dist-auth-request@w3.org, the DAV:lockdiscovery property should
                include an element DAV:principal-URL with the semantics of the
                WebDAV/ACL specification. This feature is switched-off
        by default as it lead to compatibility problems with MacOS X client.
            </description>
        </init-param>
        <init-param>
            <param-name>updateLastModified</param-name>
            <param-value>true</param-value>
            <description>
                This parameter controls whether modifying properties via
                PROPPATCH causes the last modification date of the
                resource to be updated or not.
```

```
            </description>
        </init-param>
        <load-on-startup>2</load-on-startup>
        <!-- Uncomment this to get authentication -->
        <!--security-role-ref>
            <role-name>root</role-name>
            <role-link>root</role-link>
        </security-role-ref>
        <security-role-ref>
            <role-name>guest</role-name>
            <role-link>guest</role-link>
        </security-role-ref>
        <security-role-ref>
            <role-name>user</role-name>
            <role-link>user</role-link>
        </security-role-ref-->
    </servlet>
    <!-- The mapping for the Slide WebDAV servlet.
       If you change the mapping so that the servlet isn't the default servlet
       anymore, make sure to set the init-parameter 'default-servlet' to
       'false'. -->
    <!-- iw: commented out <servlet-mapping>
        <servlet-name>webdav</servlet-name>
        <url-pattern>/webdav</url-pattern> --><!-- iw: chaged form / to
/webdav--><!--
    </servlet-mapping> -->
    <!-- iw: added new mapping -->
    <servlet-mapping>
      <servlet-name>webdav</servlet-name>
      <url-pattern>/content</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
      <servlet-name>webdav</servlet-name>
      <url-pattern>/content/</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
      <servlet-name>webdav</servlet-name>
      <url-pattern>/content/*</url-pattern>
    </servlet-mapping>
    <!-- iw: added new mapping ends -->

    <!-- For some app servers (Tomcat)
         .jsp files must be mapped
         explicitly -->
    <!--servlet-mapping>
        <servlet-name>webdav</servlet-name>
        <url-pattern>/content/*.jsp</url-pattern> --><!-- iw: changed from *.jsp to
/content/*.jsp --><!--
    </servlet-mapping-->

    <!-- Establish the default MIME type mappings -->
    <mime-mapping>
        <extension>txt</extension>
        <mime-type>text/plain</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>html</extension>
        <mime-type>text/html</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>htm</extension>
        <mime-type>text/html</mime-type>
```

```
</mime-mapping>
<mime-mapping>
    <extension>gif</extension>
    <mime-type>image/gif</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>jpg</extension>
    <mime-type>image/jpeg</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>jpe</extension>
    <mime-type>image/jpeg</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>jpeg</extension>
    <mime-type>image/jpeg</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>java</extension>
    <mime-type>text/plain</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>body</extension>
    <mime-type>text/html</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>rtx</extension>
    <mime-type>text/richtext</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>tsv</extension>
    <mime-type>text/tab-separated-values</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>etx</extension>
    <mime-type>text/x-setext</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>ps</extension>
    <mime-type>application/x-postscript</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>class</extension>
    <mime-type>application/java</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>csh</extension>
    <mime-type>application/x-csh</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>sh</extension>
    <mime-type>application/x-sh</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>tcl</extension>
    <mime-type>application/x-tcl</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>tex</extension>
    <mime-type>application/x-tex</mime-type>
</mime-mapping>
<mime-mapping>
```

```
        <extension>texinfo</extension>
        <mime-type>application/x-texinfo</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>texi</extension>
        <mime-type>application/x-texinfo</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>t</extension>
        <mime-type>application/x-troff</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>tr</extension>
        <mime-type>application/x-troff</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>roff</extension>
        <mime-type>application/x-troff</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>man</extension>
        <mime-type>application/x-troff-man</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>me</extension>
        <mime-type>application/x-troff-me</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>ms</extension>
        <mime-type>application/x-wais-source</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>src</extension>
        <mime-type>application/x-wais-source</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>zip</extension>
        <mime-type>application/zip</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>bcpio</extension>
        <mime-type>application/x-bcpio</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>cpio</extension>
        <mime-type>application/x-cpio</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>gtar</extension>
        <mime-type>application/x-gtar</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>shar</extension>
        <mime-type>application/x-shar</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>sv4cpio</extension>
        <mime-type>application/x-sv4cpio</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>sv4crc</extension>
        <mime-type>application/x-sv4crc</mime-type>
```

```
</mime-mapping>
<mime-mapping>
    <extension>tar</extension>
    <mime-type>application/x-tar</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>ustar</extension>
    <mime-type>application/x-ustar</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>dvi</extension>
    <mime-type>application/x-dvi</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>hdf</extension>
    <mime-type>application/x-hdf</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>latex</extension>
    <mime-type>application/x-latex</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>bin</extension>
    <mime-type>application/octet-stream</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>oda</extension>
    <mime-type>application/oda</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>pdf</extension>
    <mime-type>application/pdf</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>ps</extension>
    <mime-type>application/postscript</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>eps</extension>
    <mime-type>application/postscript</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>ai</extension>
    <mime-type>application/postscript</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>rtf</extension>
    <mime-type>application/rtf</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>nc</extension>
    <mime-type>application/x-netcdf</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>cdf</extension>
    <mime-type>application/x-netcdf</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>cer</extension>
    <mime-type>application/x-x509-ca-cert</mime-type>
</mime-mapping>
<mime-mapping>
```

```
        <extension>exe</extension>
        <mime-type>application/octet-stream</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>gz</extension>
        <mime-type>application/x-gzip</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>Z</extension>
        <mime-type>application/x-compress</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>z</extension>
        <mime-type>application/x-compress</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>hqx</extension>
        <mime-type>application/mac-binhex40</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>mif</extension>
        <mime-type>application/x-mif</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>ief</extension>
        <mime-type>image/ief</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>tiff</extension>
        <mime-type>image/tiff</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>tif</extension>
        <mime-type>image/tiff</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>ras</extension>
        <mime-type>image/x-cmu-raster</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>pnm</extension>
        <mime-type>image/x-portable-anymap</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>pbm</extension>
        <mime-type>image/x-portable-bitmap</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>pgm</extension>
        <mime-type>image/x-portable-graymap</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>ppm</extension>
        <mime-type>image/x-portable-pixmap</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>rgb</extension>
        <mime-type>image/x-rgb</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>xbm</extension>
        <mime-type>image/x-xbitmap</mime-type>
```

```
</mime-mapping>
<mime-mapping>
    <extension>xpm</extension>
    <mime-type>image/x-xpixmap</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>xwd</extension>
    <mime-type>image/x-xwindowdump</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>au</extension>
    <mime-type>audio/basic</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>snd</extension>
    <mime-type>audio/basic</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>aif</extension>
    <mime-type>audio/x-aiff</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>aiff</extension>
    <mime-type>audio/x-aiff</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>aifc</extension>
    <mime-type>audio/x-aiff</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>wav</extension>
    <mime-type>audio/x-wav</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>mpeg</extension>
    <mime-type>video/mpeg</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>mpg</extension>
    <mime-type>video/mpeg</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>mpe</extension>
    <mime-type>video/mpeg</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>qt</extension>
    <mime-type>video/quicktime</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>mov</extension>
    <mime-type>video/quicktime</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>avi</extension>
    <mime-type>video/x-msvideo</mime-type>
</mime-mapping>
<mime-mapping>
    <extension>movie</extension>
    <mime-type>video/x-sgi-movie</mime-type>
</mime-mapping>
<mime-mapping>
```

```
        <extension>avx</extension>
        <mime-type>video/x-rad-screenplay</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>wrl</extension>
        <mime-type>x-world/x-vrml</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>mpv2</extension>
        <mime-type>video/mpeg2</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>jpg</extension>
        <mime-type>image/pjpeg</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>sgml</extension>
        <mime-type>text/sgml</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>sgm</extension>
        <mime-type>text/sgml</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>css</extension>
        <mime-type>text/css</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>png</extension>
        <mime-type>image/png</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>bmp</extension>
        <mime-type>image/bmp</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>mpga</extension>
        <mime-type>audio/mpeg</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>mp2</extension>
        <mime-type>audio/mpeg</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>mp3</extension>
        <mime-type>audio/mpeg</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>js</extension>
        <mime-type>application/x-javascript</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>xml</extension>
        <mime-type>text/xml</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>xls</extension>
        <mime-type>application/vnd.ms-excel</mime-type>
    </mime-mapping>
    <mime-mapping>
        <extension>ppt</extension>
        <mime-type>application/vnd.ms-powerpoint</mime-type>
```

```
    </mime-mapping>
    <mime-mapping>
        <extension>doc</extension>
        <mime-type>application/msword</mime-type>
    </mime-mapping>
    <!-- Establish the default list of welcome files -->
    <!-- iw: no need to provide this in this module welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
    </welcome-file-list-->
    <!-- Authetication for the WebDAV servlet -->
    <!-- Uncomment this to get authentication -->
    <!--security-constraint>
        <web-resource-collection>
            <web-resource-name>DAV resource</web-resource-name>
            <url-pattern>/*</url-pattern>
            <http-method>COPY</http-method>
            <http-method>DELETE</http-method>
            <http-method>GET</http-method>
            <http-method>HEAD</http-method>
            <http-method>LOCK</http-method>
            <http-method>MKCOL</http-method>
            <http-method>MOVE</http-method>
            <http-method>OPTIONS</http-method>
            <http-method>POST</http-method>
            <http-method>PROPFIND</http-method>
            <http-method>PROPPATCH</http-method>
            <http-method>PUT</http-method>
            <http-method>UNLOCK</http-method>
            <http-method>VERSION-CONTROL</http-method>
            <http-method>REPORT</http-method>
            <http-method>CHECKIN</http-method>
            <http-method>CHECKOUT</http-method>
            <http-method>UNCHECKOUT</http-method>
            <http-method>MKWORKSPACE</http-method>
            <http-method>UPDATE</http-method>
            <http-method>LABEL</http-method>
            <http-method>MERGE</http-method>
            <http-method>BASELINE-CONTROL</http-method>
            <http-method>MKACTIVITY</http-method>
            <http-method>ACL</http-method>
            <http-method>SEARCH</http-method>
            <http-method>BIND</http-method>
            <http-method>UNBIND</http-method>
            <http-method>REBIND</http-method>
            <http-method>SUBSCRIBE</http-method>
            <http-method>UNSUBSCRIBE</http-method>
            <http-method>POLL</http-method>
            <http-method>NOTIFY</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>root</role-name>
            <role-name>guest</role-name>
            <role-name>user</role-name>
        </auth-constraint>
    </security-constraint>
    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>Slide DAV Server</realm-name>
    </login-config>
    <security-role>
```

```
        <role-name>root</role-name>
    </security-role>
    <security-role>
        <role-name>guest</role-name>
    </security-role>
    <security-role>
        <role-name>user</role-name>
    </security-role-->
<!-- MODULE:END org.apache.slide 1.0 -->
<!-- MODULE:BEGIN com.idega.slide 1.0 -->


    <!-- The IWSlideAuthenticator Filter -->
    <filter>
        <filter-name>IWSlideAuthenticator</filter-name>
<filter-class>com.idega.slide.authentication.IWSlideAuthenticator</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>IWSlideAuthenticator</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

<!-- MODULE:END com.idega.slide 1.0 -->
<!-- MODULE:BEGIN com.idega.setup 1.0 -->


    <!-- idegaWeb setup URL: -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/setup/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/setup</url-pattern>
    </servlet-mapping>

<!-- MODULE:END com.idega.setup 1.0 -->
<!-- MODULE:BEGIN com.idega.block.media 1.0 -->


  <servlet>
    <servlet-name>MediaServlet</servlet-name>
    <display-name>MediaServlet</display-name>
    <servlet-class>com.idega.block.media.servlet.MediaServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>MediaServlet</servlet-name>
    <url-pattern>/servlet/MediaServlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>MediaServlet</servlet-name>
    <url-pattern>/servlet/MediaServlet/</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>MediaServlet</servlet-name>
    <url-pattern>/servlet/MediaServlet/*</url-pattern>
  </servlet-mapping>

<!-- MODULE:END com.idega.block.media 1.0 -->
```

```
<!-- MODULE:BEGIN com.idega.builder 1.0 -->


  <servlet>
    <servlet-name>IBMainServlet</servlet-name>
    <display-name>IBMainServlet</display-name>
    <servlet-class>com.idega.builder.servlet.IBMainServlet</servlet-class>
    <load-on-startup>-2</load-on-startup>
  </servlet>
  <servlet>
    <servlet-name>IBIFrameServlet</servlet-name>
    <display-name>IBIFrameServlet</display-name>
    <servlet-class>com.idega.builder.servlet.IBIFrameServlet</servlet-class>
    <load-on-startup>-1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/builder</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/builder/</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/builder/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/IBMainServlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/IBMainServlet/</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>IBMainServlet</servlet-name>
    <url-pattern>/servlet/IBMainServlet/*</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>IBIFrameServlet</servlet-name>
    <url-pattern>/servlet/IBIFrameServlet</url-pattern>
  </servlet-mapping>

<!-- MODULE:END com.idega.builder 1.0 -->

</web-app>
```

1.15 Examples Of LDAP property settings javaldap.prop

LDAP Properties Settings

```
#JavaLDAP Server Properties
#Sat Jun 10 11:34:38 CDT 2000

# Basic Server Identification
javaldap.server.name=localhost
javaldap.server.port=10389

#IdegaWeb auto start embedded ldap server on startup
idegaweb.ldap.autostart=false

#IdegaWeb IWLDAPWS.jws webservice port, the same as the http port of the server
idegaweb.webservice.port=80

# Not yet used...plan to pool threads soon
javaldap.server.threads=16

# Schema file locations...Only STD is currently used
# These files are in DSML format
javaldap.schema.user=./user.oc.xml
javaldap.schema.std=./std.oc.xml

# Backend Configuration
javaldap.server.backends=./backends.prop

# Schema Checking On=1/Off=0
javaldap.schemacheck=0

# ACL Checking On=1/Off=0
javaldap.aclcheck=0
javaldap.acl.props=./acls.prop

# Root User and Password - Bypass ACLs
javaldap.rootuser=cn=Admin
javaldap.rootpw=manager

# Debug Level (Currently 0-9 with 9 being most verbose)
javaldap.debug=9

# JDBC Config for HSQL
#javaldap.backendjdbc.longvarchar=LONGVARCHAR
#javaldap.backendjdbc.createtable=CREATE TABLE
#javaldap.backendjdbc.dbdriver=org.hsql.jdbcDriver
#javaldap.backendjdbc.dburl=jdbc:HypersonicSQL:jldapdb
#javaldap.backendjdbc.dbuser=sa
#javaldap.backendjdbc.dbpass=

# JDBC Config for IBM DB2 - be sure that the database 'jldapdb' (or whatever
#     you change it to) exists before running. Tables will be configured.
#javaldap.backendjdbc.longvarchar=LONG VARCHAR
#javaldap.backendjdbc.createtable=CREATE TABLE
#javaldap.backendjdbc.dbdriver=COM.ibm.db2.jdbc.app.DB2Driver
#javaldap.backendjdbc.dburl=jdbc:db2:jldapdb
#javaldap.backendjdbc.dbuser=db2admin
```

```
#javaldap.backendjdbc.dbpass=manager

# JDBC Config for InstantDB
#javaldap.backendjdbc.longvarchar=LONGVARBINARY
#javaldap.backendjdbc.createtable=CREATE TABLE
#javaldap.backendjdbc.dbdriver=org.enhydra.instantdb.jdbc.idbDriver
#javaldap.backendjdbc.dburl=jdbc:idb:jldap-instantdb.prp
#javaldap.backendjdbc.dbuser=
#javaldap.backendjdbc.dbpass=


# JDBC Config for Oracle 8i
#
#javaldap.backendjdbc.longvarchar=LONG VARCHAR
#javaldap.backendjdbc.createtable=CREATE TABLE
#javaldap.backendjdbc.dbdriver=oracle.jdbc.driver.OracleDriver
#javaldap.backendjdbc.dburl=jdbc:oracle:oci8:@
#javaldap.backendjdbc.dbuser=test
#javaldap.backendjdbc.dbpass=tester
```