

## **Join v.1.0**

### **Project Documentation**



# Table of Contents

---

<b>1 About Join</b>	
1.1 What is Join ?	1
1.2 Feature Summary	2
1.3 Downloading	4
1.4 Release Notes	5
<b>2 Getting Started</b>	
2.1 Installing	9
2.1.1 Standard Setup	13
2.1.2 Custom Setup	14
2.2 Configuring a Project	19
2.3 FAQ	25
<b>3 User's Guide</b>	
3.1 Artifacts	26
3.2 Environments	29
3.3 Workflow	30
3.3.1 Dep. Lifecycle	31
3.4 Integrator's Guide	32
3.4.1 Repositories	33
3.4.2 Domain Properties	35
3.4.3 User Access	39
3.4.3.1 User Definitions	40
3.4.4 Scripted Process	41
3.4.4.1 Ant Scripts	44
3.4.4.2 Groovy Scripts	47
3.4.5 Remote Access	48
3.4.5.1 SOAP	49
3.4.5.2 Hessian	50
3.4.5.3 XML-RPC	51
3.4.6 Setup Modes	52

<b>4 Related Projects</b>	
4.1 Ant. ....	54
4.2 Spring. ....	55
4.3 ActiveMQ. ....	56

## 1.1 What is Join ?

---

## 1.2 Feature Summary

---

### Feature Summary

Join has an extensive list of features. The following are the key one described in a nutshell:

#### Artifacts Management

This feature of Join allows you to manage all the artifacts of your software project: from the delivery of development teams to the production-ready build. It is able to connect to **Software Configuration Management** tools to collect promoted components and publish all the artifacts onto **repositories** (Structured file system, FTP or WebDAV implementations at time of writing).

#### Environments Configuration Management

Join allows you to **specify** your test and validation environments in terms of resources, services gateways and physical machines. It acts as a Configuration Management Tool for your environments by letting you define your own elements and **track configuration changes** on the environments elements.

#### Workflow & Automation

Join application defines a minimal workflow that can be adapted to your own project **Integration Process**. The workflow transitions acts as event producers that can be asynchronously processed. Join is made to easily integrates triggers for these events by providing adapters for script languages such as **Ant**, **Javascript** or **Groovy**. Therefore, web frontend actions can be translated into scripts execution for **automating** your software builds or deployments.

#### Communication

Join can also be seen as a communication tool, allowing teams to be up-to-date with project integration events. Information diffusion can be done through the portal interface, through **mailing lists** with a subscription system on fine grained events or through **RSS** feeds. Every workflow event can be published through these means.

#### Integration

Integration with other enterprise software is possible through the use of **Web Services**. Many Join services are exposed as **SOAP**, **Hessian** or **XML-RPC** web services. An example of integration use-case is an Eclipse plugin allowing developer to deliver its work into Join from the IDE (Check [here](#) for this work in progress).

#### Database Agnostic

Join uses **Hibernate** as the ORM tool for the persistence layer which enables pluggable databases (Hypersonic, MySQL, PostgreSQL, and Oracle tested at time of writing). This allows your organization

to leverage existing resources without having to purchase new database hardware and software.

### **Personalization**

Join leverages off of **Struts** and **Tiles** frameworks to follow the MVC pattern. The look and feel of the application can be easily customized and reskinned because the display logic is concentrated in a few template files read by Tiles. The web interface supports **internationalization** through the usage of Struts message resources bundles. Language resources for English, French and Spanish are already provided.

### **Scalability**

Join is very scalable in many ways. Its architecture has been designed with scalability in mind.

First, it allows you to deploy web application and workflow scripts - running asynchronously - on **separate machines**. Thus, you may have dozens of scripts triggered by a workflow event without affecting the response time of the server running web application.

The web application itself is made to be clusterable and uses a pluggable cache provider architecture. You can scale by adding more nodes without sacrificing on caching.

Finally, the asynchronous part running scripts is also scalable thanks to the high availability configuration options of **ActiveMQ**, the JMS provider used for event publication.

## 1.3 Downloading

---

### Download a Join distribution

Join is currently distributed in several formats for your convenience. For instructions on how to install Join, see the [Installation Instructions](#).

#### Stable release: Join 1.1

Join 1.1 is our latest final stable release available since **2008-02-04**. Thank you for downloading and trying out Join. Feel free to send us feedback on the mailing lists or forums.

You can download the standalone bundle for JDK 1.4 here:

- [.zip archive](#)
- [.tar.gz archive](#)

You can download the standalone bundle for JDK 1.5 here:

- [.zip archive](#)
- [.tar.gz archive](#)

You can download the WAR bundle here:

- [.zip archive](#)
- [.tar.gz archive](#)


Join is distributed under the [General Public License, version 2.0](#).











## 1.4 Release Notes



### Release History

Version	Date	Description
<a href="#">1.1</a>	2008-02-04	1.1 release
<a href="#">1.0</a>	2007-04-10	1.0 release
<a href="#">1.0-rc2</a>	2007-03-01	Second release candidate for 1.0
<a href="#">1.0-rc1</a>	2007-02-06	First release candidate for 1.0
<a href="#">1.0-20070111-SNAPSHOT</a>	2007-01-11	Third snapshot for 1.0
<a href="#">1.0-20061011-SNAPSHOT</a>	2006-10-11	Second snapshot for 1.0




Get the RSS feed of the last changes 

### Release 1.1 - 2008-02-04






Type	Changes	By
	Ajax support when navigating from left menu and in displaytag tables when sorting columns.	<a href="#">lbroudoux</a>
	Remote service definition allowing to interact with resource version creation and resource update. This will allow a third party system to notify Join when updating a managed resource.	<a href="#">lbroudoux</a>
	Mailing list subscriptions were not displayed. Fixes <a href="#">1880908</a> .	<a href="#">lbroudoux</a>
	Deployments history were not displayed for LogicalEnvironment, PhysicalEnvironment and EnvironmentMappings. Links were broken and retrieval not fully implemented (return partial results). Fixes <a href="#">1821986</a> .	<a href="#">lbroudoux</a>
	Fix the redirection URL in case of missing field into login form. Fixes <a href="#">1858374</a> .	<a href="#">lbroudoux</a>
	Fix the component type description mandatory check that was incoherent with web interface. Add the display of description within build creation form. Fixes <a href="#">1821817</a> .	<a href="#">lbroudoux</a>
	Fix the setup steps numbers, labels and display width when installing in custom setup mode. Fixes <a href="#">1834746</a> .	<a href="#">lbroudoux</a>
	Fix the white screen issue when browsing empty assemblies collection. Add a test to item presence before trying to invoke the getStatus() method on it. Fixes <a href="#">1817060</a> .	<a href="#">lbroudoux</a>

Type	Changes	By
	Fix the custom setup process that is using an external database. It was failing because of a column length constraint onto Event entity that was not respected. Issue does not apply on Hypersonic and that's why this regression was not detected ;( Fixes <a href="#">1834745</a> .	<a href="#">lbroudoux</a>
	Remove a bunch of Checkstyle errors (nearly 200).	<a href="#">lbroudoux</a>





## Release 1.0 - 2007-04-10

Type	Changes	By
	Support for getting environment configuration view for a specified date.	<a href="#">lbroudoux</a>
	Actually active the access control filter for security constraints check.	<a href="#">lbroudoux</a>
	Remove a bunch of Checkstyle errors (nearly 100).	<a href="#">lbroudoux</a>











## Release 1.0-rc2 - 2007-03-01

Type	Changes	By
	Support for actually binding EIS to gateway and providing a gateway configuration changes view.	<a href="#">lbroudoux</a>
	Add an access control filter that guarantees the respect of security constraints. This filter operates on every incoming requests and uses the join-access-control.xml file as the configuration backend.	<a href="#">lbroudoux</a>
	Add a PasswordEncoder implementation using a MD5 digest for passwords. This provides an example on how to integrate with security constrained user definition sources.	<a href="#">jevrard</a>
	Fix the remote service allowing to get event consumer infos. This was an Hibernate lazy loading problem solved by adding an HibernateInterceptor (opens Session and stick to current Thread when invoking a service). Fixes <a href="#">1667133</a> .	<a href="#">lbroudoux</a>
	Fix a bad link on deployment parameter values management page. Fixes <a href="#">1667123</a> .	<a href="#">lbroudoux</a>











## Release 1.0-rc1 - 2007-02-06

Type	Changes	By
	Support for notifications triggered by events (mail and Jabber/XMPP protocols supported)	<a href="#">lbroudoux</a>
	Add UserDao implementation using raw Jdbc. This provides an example on how to integrate Join with external user definition sources and allow also many Join deployed applications to share the same user definitions.	<a href="#">lbroudoux</a>
	Fix the password management in case of remote services call (when synchronous side of Join calls web services from asynchronous side).	<a href="#">lbroudoux</a>
	Update the view of existing User profile. Allow user to manage their account by changing password, personal informations and subscribe to mailing lists.	<a href="#">lbroudoux</a>

## Release 1.0-20070111-SNAPSHOT - 2007-01-11

Type	Changes	By
	Support Oracle as available database	<a href="#">lbroudoux</a>
	ComponentTypes management operations support.	<a href="#">lbroudoux</a>
	Components management operations support.	<a href="#">lbroudoux</a>
	Builds management operations support.	<a href="#">lbroudoux</a>
	ResolveUserTag was not correctly releasing resources on exiting. This may lead on inaccurate user's label when having no user (ex: no assignee on a deployment). The displayed value was previous resolution success because of JSP tags reuse. Fixes <a href="#">1630759</a> .	<a href="#">lbroudoux</a>
	Missing labels. Fixes <a href="#">1613499</a> .	<a href="#">lbroudoux</a>
	Shutdown does not stop JMSConsumers. Fixes <a href="#">1613501</a> .	<a href="#">lbroudoux</a>
	MessageService throws InvalidSessionException. Fixes <a href="#">1613498</a> .	<a href="#">lbroudoux</a>
	NullPointerException in QuartzCronActions class. Fixes <a href="#">1606790</a> .	<a href="#">lbroudoux</a>
	MalformedURLException in CronActions. Fixes <a href="#">1606680</a> .	<a href="#">lbroudoux</a>

## Release 1.0-20061011-SNAPSHOT - 2006-10-11

Type	Changes	By
	Add a Xml-Rpc interface for remote services access.	<a href="#">lbroudoux</a>
	Add a page for displaying environment configuration changes that occurred during a time period.	<a href="#">lbroudoux</a>
	Added deployment parameters management. Deployment parameter values can be expressed using other parameter or using environment configuration information (through metadata with Ant style notation).	<a href="#">lbroudoux</a>
	Release cache is now refreshed from datastore when a release update fails. Fixes <a href="#">1567538</a> .	<a href="#">lbroudoux</a>
	Step cache is now refreshed from datastore each time cascade updates occurs. This avoid having Hibernate to work from stale cached objects and failing synchronizing them. Fixes <a href="#">1567529</a> .	<a href="#">lbroudoux</a>
	Fix the position shifting algorithm of step objects. Fixes <a href="#">1567530</a> .	<a href="#">lbroudoux</a>
	Corrected a bug that prevent from registering consumer parameters.	<a href="#">lbroudoux</a>
	Corrected properties produced by the PhysicalEnvironmentPropertiesExtractor. Machine related properties are now correctly spelled.	<a href="#">lbroudoux</a>
	Asynchronous message consumers can now be deactivated without a servlet container restart. Deactivation method is now called on consumer activation property update.	<a href="#">lbroudoux</a>
	Improve navigation through web interface using transversal hyperlinks. Left menu is less used than in previous snapshot.	<a href="#">lbroudoux</a>

## 2.1 Installing

---

### Choosing a distribution

Join is shipped into 2 distributions : [Join Standalone](#) which runs out of the box; and [Join WAR](#) which is deployable as a Web Archive in any J2EE application server or servlet container.

The **Join Standalone** distribution is bundled with an application server (Apache Tomcat) and a database (Hypersonic SQL). It is recommended for : first-time users, evaluators, users without an already running application server.

The **Join WAR** distribution requires a J2EE application serveran knowledge on how to configure and use it. It is ideal for : Java veterans and users wanting to deploy onto existing application server.

### Join Standalone

#### Requirements

##### System

- Any operating system supporting J2EE 1.4+, including Windows, Linux, Mac OS X, Solaris, AIX.
- Web browser with cookies enabled (IE5+, Mozilla is recommended).
- 1 GHz or higher Pentium 4, or equivalent (2.4GHz Pentium Xeon or equivalent recommended)
- 256MB RAM minimum (512MB+ recommended)
- 50MB of free disk space (disk usage may be greater depending on how you setup Join but 50MB is really enough for starting up).

##### Databases

Join is configured to run with its own embedded database. However, this is recommended only for evaluation or demonstration. To ensure your data is kept safe and consistent, we recommend production deployments using one of these external databases:

- MySQL 5.0
- PostgreSQL 8.1
- Oracle 8i, 9i

##### Others

- Sun JDK 1.4 and above (other 1.4+ JDKs can also be used, but require the Sun JSSE package).  
Download the J2SE SDK version without the Sun Java System Application Server.

### Installing Standalone distribution

#### Step One. Installing the Java Development Kit

If you have already installed the Java Developers Kit (JDK), you may skip to Step Two. You can check this by opening a command prompt (Start -> Run -> type 'cmd' and hit enter), and then entering the following:

```
echo %JAVA_HOME%
```

On machines on which the JDK is installed, a directory will be printed ending in `jdk<VERSION>`, eg `C:\Program Files\Java\jdk1.4.2_07` on Windows. However if it ends in `jre<VERSION>`, or if nothing is printed:

1. Download the JDK [here](#) and run the installer.
2. Set the `JAVA_HOME` environment variable to the directory in which Java is installed.

### Step Two. Downloading Join

Download the standalone bundle of Join from this [page](#).

### Step Three. Running the Setup Sizard

1. Go to your `<INSTALL>` directory and open `<INSTALL>/join/WEB-INF/classes`. Edit the file named `join-init.properties` and find the line

```
join.home=c:/join/home
```

Change it to make it point to the Join home directory. The home directory is where Join will store

2. Go to `<INSTALL>/tomcat/` and run the startup script. Windows users should open a command prompt and run on `<INSTALL>\tomcat\startup.bat` (so that they can view any error messages), while users of Unix variants run `<INSTALL>/bin/startup.sh`.
3. Go to your web browser and enter the address <http://localhost:8080/join>. The Join Setup Wizard should appear, proceed to Setting Up Join.

## Join WAR

### Requirements

#### System

- Any operating system supporting J2EE 1.4+, including Windows, Linux, Mac OS X, Solaris, AIX.
- Web browser with cookies enabled (IE5+, Mozilla is recommended).
- 1 GHz or higher Pentium 4, or equivalent (2.4GHz Pentium Xeon or equivalent recommended)
- 256MB RAM minimum (512MB+ recommended)
- 50MB of free disk space (disk usage may be greater depending on how you setup Join but 50MB is really enough for starting up).

#### Application Servers

- Tomcat (5.0 and above)
- JBoss (3.2 and above)
- Resin (3.0 and above)
- Apache Geronimo (1.0)
- Glassfish (1.0)

- Other J2EE compliant servers

### Databases

Join is configured to run with its own embedded database. However, this is recommended only for evaluation or demonstration. To ensure your data is kept safe and consistent, we recommend production deployments using one of these external databases:

- MySQL 5.0
- PostgreSQL 8.1
- Oracle 8i, 9i

### Others

- Sun JDK 1.4 and above (other 1.4+ JDKs can also be used, but require the Sun JSSE package).  
Download the J2SE SDK version without the Sun Java System Application Server.

## Installing WAR distribution

### Step One. Download and extract WAR

Download the WAR bundle of Join from this [page](#).

### Step Two. Configure join-init.properties

### Step Three. Add a new context into Application Server

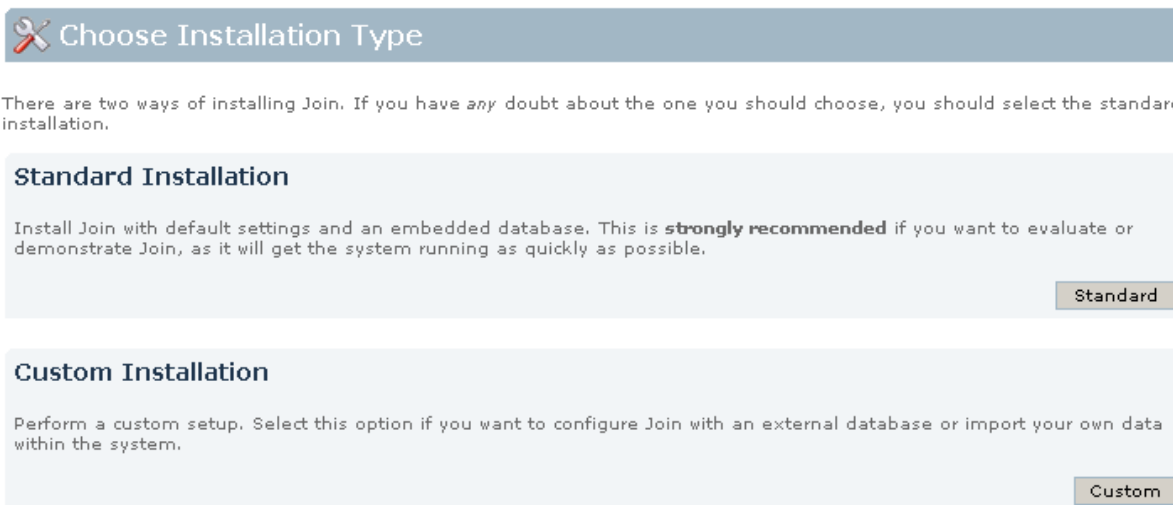
### Step Four. Running the Setup Wizard

Go to your web browser and enter the address corresponding to the context you have created into your Application Server. The Join Setup Wizard should appear, proceed to Setting Up Join.

## Setup Wizard

There are 2 ways of installing Join. Depending on the setup type you choose, continue this guide on [Standard Setup](#) documentation page or on [Custom Setup](#) page.

*Screenshot: Choosing setup type*



**Choose Installation Type**

There are two ways of installing Join. If you have *any* doubt about the one you should choose, you should select the standard installation.

**Standard Installation**

Install Join with default settings and an embedded database. This is **strongly recommended** if you want to evaluate or demonstrate Join, as it will get the system running as quickly as possible.

**Standard**

**Custom Installation**

Perform a custom setup. Select this option if you want to configure Join with an external database or import your own data within the system.

**Custom**





## 2.1.1 Standard Setup

---

### Standard Installation

The **Standard Installation** is recommended for users evaluating Join features. It comes with its embedded database and allows you to start working within only two clicks.

In this step, you will define Join's first registered user. This user will have administrative privileges over the entire Join installation. After you've finished setup, you will use this user to add other users and to assign them permission to access various parts of the Join site. In the form displayed, enter a username, password, last and firstname an administrator account.

You are now ready to start using Join !

*Screenshot: Setting up Join administration account*

The screenshot shows a web-based setup interface. On the left, a 'Setup Steps' sidebar lists three steps: '1. Standard Installation' (checked), '2. Create administrator' (active), and '3. Setup is complete' (marked with a red X). The main area is titled 'Setup Join Administrator' and contains the instruction 'Please configure the administrator account for this installation of Join.' Below this is a 'Configure account' section with five input fields: 'Login', 'Password', 'Confirmation', 'Lastname', and 'Firstname'. The 'Login' field is bolded. A 'Next >>' button is located at the bottom right of the form.

The form fields in **bold** represents mandatory informations.

## 2.1.2 Custom Setup

---

### Custom Installation

The **Custom Installation** is recommended for users already knowing Join internal mechanism and or wanting to use an external database. It allows you to choose your installation mode and configure the connection settings to database.

This installation process is not linear: depending on your choices the steps described here in subsections may be required or not, ordered or not so that the progression bar on the left may have different bullets. For simplicity, we recommend you to follow the navigation links at the end of each step.

The first choice to take is whether you want to install Join on a **single machine** (the one hosting the Join web application) or whether you want to realize installation on two machines thus **dissociating** asynchronous processing (scripts) from synchronous services (web application).

*Screenshot: Choosing single/dissociated installation*

**Setup Steps**

- ✓ 1. Custom Installation
- ✓ 2. Choose Dissociation
- ✗ 3. ...
- ✗ 4. Create administrator
- ✗ 5. Setup is complete

**Choose Installation Mode**

2 installation modes are possible depending on the fact that you want to dissociate *Asynchronous* processing from *Synchronous* services.

**Single installation**

Choose this option if you want Join to be installed on a single machine (the one that is hosting your Web server).

**Simple**

**Dissociated installation**

Choose this option if you want to dissociate **Asynchronous** processing from **Synchronous** services on different machines. You should install the asynchronous processing side of Join **first** then the synchronous services side.

**Dissociated**

Go to [Database configuration](#) step if you have chosen a **Single installation** or go to [Application side choice](#) step if you have chosen a **Dissociated installation**.

### Installation Steps

#### Database configuration

*Screenshot: Choosing embedded/external database*

 Choose Installation Type

Choose database configuration.

**Embedded database.**

This option allows Join to operate without an external database. This is **recommended for evaluating or demonstrating Join**, but a production system should use an external database for improved scalability and reliability.

**External database.**

This option allows you to configure your own external database. You may choose it from a list of supported databases. This is **recommended for production systems**.

Go to [Configuring administrator account](#) step if you have chosen an **Embedded database** or go to [Configuring external database/datasource](#) step if you have chosen an **External database**.

**External database**


You may choose a direct database access configuration or a database access through a JNDI datasource configured onto the application server Join is running in. These two options need a common parameter that is the kind of database you're connection to (so you can tell Join which "dialect" it needs to speak).

The first option uses a standard JDBC database connection. Connection pooling is handled within Join. You will need to know:

- The Java class name for the appropriate database driver.
- The JDBC URL for the database you will be connecting to.
- A valid username and password for that database.
- The size of the connection pool Join should maintain (If in doubt, just go with the default provided).

The second option required that you have configured a datasource into your application server, and know its JNDI name. Note, some servers will have JNDI names like jdbc/datasourcename, others will be of the form java:comp/env/jdbc/datasourcename. Consult your server documentation.

*Screenshot: Configuring external database/datasource*

 Database configuration

Please fill the form.

**Common parameter**  
Database Type :   
Dialect :

**Database configuration**  
Driver Class Name :   
Database URL :   
User Name :   
Password :   
Pool Size :

**Datasource configuration**  
Datasource Name :

The form fields in **bold** represents mandatory informations.

Next and last step is [Configuring administrator account](#) step.

### Application side choice

Now that you have chosen a dissociated installation, you have to tell the setup wizard whether you are now installing the synchronous services side or the asynchronous processing side of the application.

Note, you first have to install the asynchronous processing side. This allows you to specify the connection parameters used by the web application to access it. Then you will have to repeat the installation, on another server, for synchronous web services side and give the access parameters specified before.

*Screenshot: Choosing application side*

## Choose services to install

Allows to distinguish the installation server being configured...  
The configuration will be different depending on the side of Join application - **Synchronous** web services or **Asynchronous** script processing services - you want to install.

### Installation of Synchronous side

Choose this option if you want to configure the server dedicated to **synchronous** web services. *Note: You should have configured asynchronous side first.*

Synchronous

### Installation of Asynchronous side

Choose this option if you want to configure the server dedicated to **asynchronous** script processing.

Asynchronous

Go to [Access to asynchronous side](#) step if you have chosen the installation of **Synchronous side** or go to [Access from synchronous side](#) step if you have chosen the installation of **Asynchronous side**.

### Access to asynchronous side

You are now installing synchronous side of Join application. You have to provide here the web URL and the messages broker URL of the asynchronous side you have previously installed. The web URL is an HTTP URL (eg. <http://asynchost.mycomp.com:8080/join>) ; the messages broker URL is something like <tcp://asynchost:61616> (depending on the port you have specified when installing asynchronous side).

*Screenshot: Configuring access to asynchronous side*

## Configuration of application other side access

This form needs to be filled with **parameters allowing to access** to Join application other side ; or being **accessed from the other side**. (Synchronous services for asynchronous side, asynchronous processing for synchronous services side).

### Access parameters

**Other side access URL :**

**Messages broker URL :**

Next >>

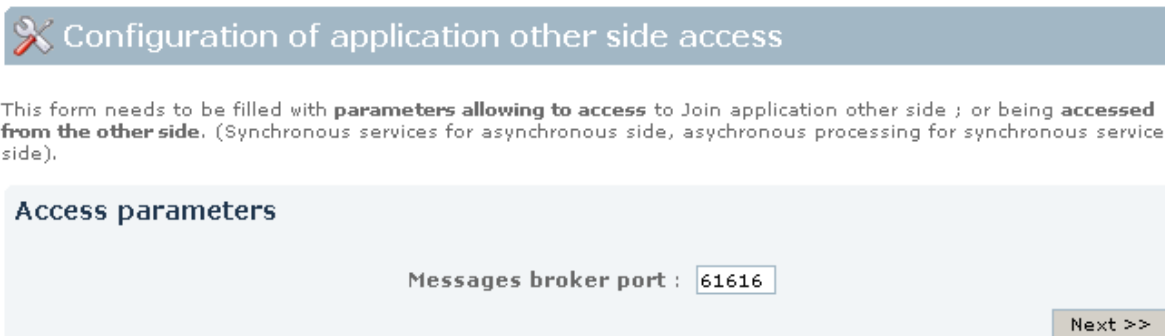
The form fields in **bold** represents mandatory informations.

Next step is [Database configuration](#) step.

### Access from synchronous side

You are now installing asynchronous side of Join application. You have to specify the listening port of the integrated messages broker in order to be reached from the synchronous side. (If in doubt, just go with the default provided).

*Screenshot: Configuring access from synchronous side*



**Configuration of application other side access**

This form needs to be filled with **parameters allowing to access** to Join application other side ; or being **accessed from the other side**. (Synchronous services for asynchronous side, asynchronous processing for synchronous services side).

**Access parameters**

Messages broker port :

Next >>

The form fields in **bold** represents mandatory informations.

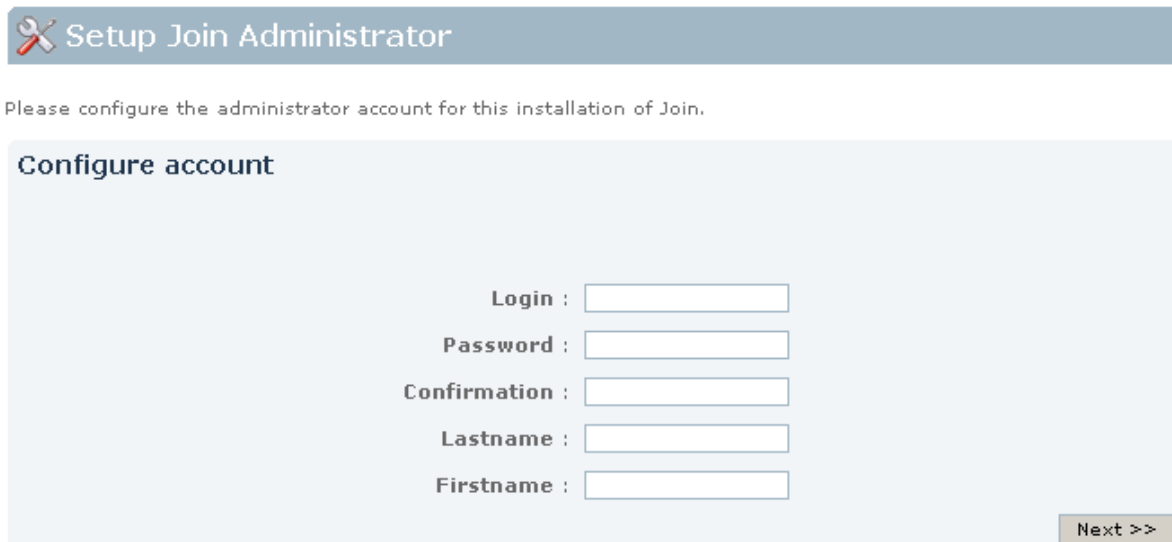
Next and last step is [Configuring administrator account](#) step.

### Configuring administrator account

This is the setup last step. In this step, you will define Join's first registered user. This user will have administrative privileges over the entire Join installation. After you've finished setup, you will use this user to add other users and to assign them permission to access various parts of the Join site. In the form displayed, enter a username, password, last and firstname for administrator account.

You are now ready to start using Join !

*Screenshot: Configuring administrator account*



**Setup Join Administrator**

Please configure the administrator account for this installation of Join.

**Configure account**

Login :

Password :

Confirmation :

Lastname :

Firstname :

Next >>

The form fields in **bold** represents mandatory informations.

## 2.2 Configuring a Project

---

### Configuring a Project

This is a quick start guide on things that should be first configured when starting using Join on your software project. This is not a complete configuration reference but just a list of things that should be done.

#### Declaring Releases

A release entity just corresponds to a **planned release** of the software project you are working on. Every entity produced or managed by Join is then associated to a specific release. A software release of your project will have a major and a minor number, a distinct name and a shipping date.

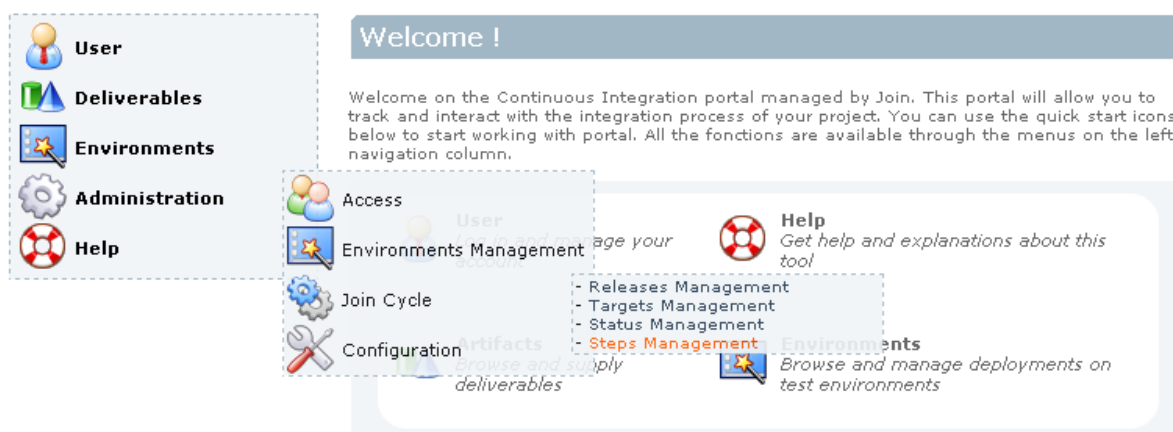
*Screenshot: Menu access for releases management*



#### Declaring Integration Process Steps

A step in Join system is a **stage** that helps defining your own integration process. There is often the need that tests may be performed at stretch by different teams (thus the need of a process). Defining **steps and their order**, allows you to later build business rules for controlling the lifecycle of your assemblies and your environment updates.

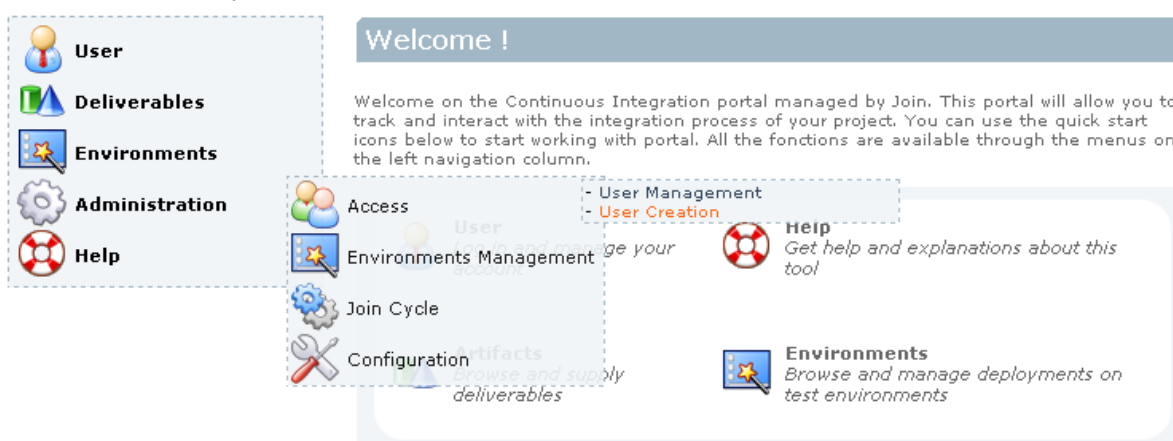
*Screenshot: Menu access for steps management*



### Declaring Deliverable Types

Deliverables are a special **category of artifacts** that are supplied by your project development teams. One of the first task into configuring Join is to declare the deliverable types corresponding to your teams. Only granted users are allowed to make this declaration, so let's begin creating a new user.

*Screenshot: Menu access for user creation*



After having created a new user similarly as in [standard setup](#), the application is asking you to assign this user some security roles. 5 default roles exist in Join:

- *admin*: have administration privileges such as creating users,
- *architect*: have permissions to define project organization such as deliverables, components and resources types,
- *joiner*: represents a member of project integration team,
- *manager*: is responsible of environment management,
- *supplier*: is allowed to supply new deliverables.

In order to declare new deliverable types, user should have the privileges associated to the *architect* and *joiner* roles. So check the boxes and update the rights of your newly created user (John Doe here).

*Screenshot: Security roles assignment*



The screenshot shows a configuration page for a user named 'Doe, John'. The page title is 'Security roles'. It contains a table with three columns: 'Role', 'Yes / No', and 'Resources'. The table lists five roles: 'admin', 'architect', 'joiner', 'manager', and 'supplier'. The 'Yes / No' column contains checkboxes, with 'architect' and 'joiner' checked. The 'Resources' column contains empty text input fields for the 'manager' and 'supplier' roles. Below the table are 'Submit' and 'Reset' buttons. At the bottom left, there is a 'New role ...' button with a plus icon.

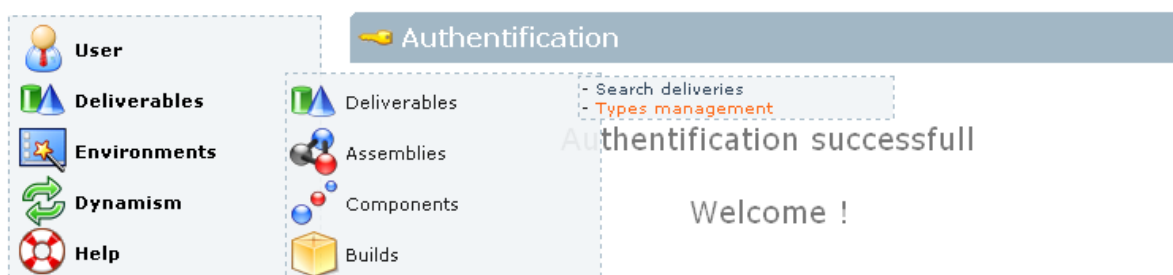
Role	Yes / No	Resources
admin	<input type="checkbox"/>	
architect	<input checked="" type="checkbox"/>	
joiner	<input checked="" type="checkbox"/>	
manager	<input type="checkbox"/>	<input type="text"/>
supplier	<input type="checkbox"/>	<input type="text"/>

Submit Reset

New role ...

Logout from application and re-enter it using you new user login. You may now be able to manage the deliverable types of your project using the Artifacts/Deliverables menu entry has shown below.

*Screenshot: Menu access for deliverable types management*



As said on the deliverable types management pages, types should have a unique identifier and a label for display. One important attribute of type is the **keys template**: a template that will be used by Join to generate unique identifiers for your deliverables. When supplying a new deliverable, the application will replace the {0} string by the name of the **release** this delivery is for and the {1} string by the version of the deliverable. For this first example, we declare a simple type that is mandatory into assembly creation and not extracted from SCM (this points will be discussed later).

*Screenshot: Deliverable type creation*

## Deliverable types

This page allows you to manage deliverable categories. This **types** help organizing your development teams works ; **deliverables** will collect them. A deliverable type has an unique **identifier**, a **label**, a **template** that helps creating deliverables identifiers and a **delivery mode** (file upload or SCM extraction).

### Managed types

Key	Label	Appear in Versions ?	Mandatory in Versions ?	Supplied through VCS ?
fwk	Java Framework	true	true	false

Exports possibles:  CSV  Excel  XML

### Type update

<b>Key :</b>	<input type="text" value="fwk"/>
<b>Label :</b>	<input type="text" value="Java Framework"/>
<b>Template for Deliverable keys :</b>	<input type="text" value="fwk-r{0}-v{1}"/>
<b>Appear in Versions ? :</b>	Yes <input checked="" type="radio"/> No <input type="radio"/>
<b>Mandatory in Versions ? :</b>	Yes <input checked="" type="radio"/> No <input type="radio"/>
<b>Supplied through VCS ? :</b>	Yes <input type="radio"/> No <input checked="" type="radio"/>

"fwk"

The form fields in **bold** represents mandatory informations.

## Making a first Delivery

As a first example on what to do once Join been configured, here's how to supply a first deliverable for your project into Join.

### Creating a Supplier

Create a user having the *supplier* security role for one deliverable type you have previously declare (see [above](#) for details).

*Screenshot: Supplier role assignment*

Role	Yes / No	Resources
admin	<input type="checkbox"/>	
architect	<input type="checkbox"/>	
joiner	<input type="checkbox"/>	
manager	<input type="checkbox"/>	
supplier	<input checked="" type="checkbox"/>	Java Framework

Submit Reset

New role ...

### Supplying a Deliverable

Logout from application and re-enter it using the new supplier account login. You may now be able to access the **delivery form** as shown in screenshot below.

*Screenshot: Menu access for making a delivery*

© 2006, Join Team

SOURCEFORGE.net

You may now fill the form to supply a new deliverable: choose the release this deliverable is for, give the version information and consign some comments on you delivery. The last field is dedicated to the **deliverable file** that will be uploaded onto Join server repository.

*Screenshot: Delivery creation form*

## Delivery

This form allows you to declare a new **delivery** for the deliverable type you can supply. A new deliverable (corresponding to his delivery) must be bound to one **release** or your software project. It should also have some information on its **version**, some **comments** and possibly a **file** corresponding to its content if deliverable is not extracted from a Software and Configuration Management tool.

### Deliverable 'Java Framework'

New delivery

**Release :**

**Version :**

**Comments :**

**Deliverable file :**

The form fields in **bold** represents mandatory informations.

As a result, you may see your delivery is now registered and appear in the list. Delivery browsing is accessible to all users so that they can keep in touch with the work done by others...

*Screenshot: Delivery browsing*

## Deliverables

This page allows to search and browse the different **deliveries** already registered into the system. Selection is done using deliverables type and release have been supplied for.

### Deliverables selection

Type :  Release :

### Selected deliverables

Key	Delivery date	Supplier
fwk-r1.0-01	2006 août 23	Sawyer Tom 

Exports possibles:  CSV  Excel  XML

 Deliver 'Java Framework' ...

## 2.3 **FAQ**

---

## 3.1 Artifacts

---

### Artifacts Concepts

One of the core feature of Join is the management of all your software project artifacts: from the delivery of the development teams to the production-ready build. Before going further into artifacts concepts let's define here what we call an artifact: *an object that is the result from a human activity or from a transformation process.*

The expected properties of artifacts are:

- easy referencement: the need of a unique identifier,
- classification: the need to categorize and type them,
- auditability: the need to know when they are created and how they are used,
- and the ability to store them "somewhere".

Another root concept related to artifacts is the one of **Release** that is a major version of the software project. Releases are usually planned and appear on software roadmaps: they represent the software itself but may also act as "container" of features to embed for shipping date. They are **related to artifacts** because each artifact is produced within the context of a release and thus is bound to it into Join.

*(Note: many other project management tools already use the concept of release or major version. As an example, check out [Atlassian JIRA roadmap report](#))*

The following sections detail the 4 categories of artifacts managed into a Join application:

- deliverables,
- assemblies,
- components,
- builds.

### Deliverable

A **Deliverable** within Join is not different from a deliverable within any software plan. It constitutes the input of the software staging workflow realized through Join. A deliverable represents an artifact supplied by a development team. A deliverable does not necessarily match a component of software to produce ; it should rather be seen as a commodity to bundle work done by a team. Therefore deliverables can be **typed** in order to suit your project Human Organization.

### Assembly

An **Assembly** within Join is kind of a "container" made using a combination of deliverables. It is composed by choosing a specific instance of deliverable within each defined **Deliverable Types** (some types may be optional). By grouping deliverables within an assembly, the assembler aims is to produce a coherent software snapshot that can be later deployed onto integration environments. The actual "creation" of assembly - that is the unpacking/packing stuffs - will allow you to dynamically discover the next kind of artifact that composing it: that is the **Components**.

## Component

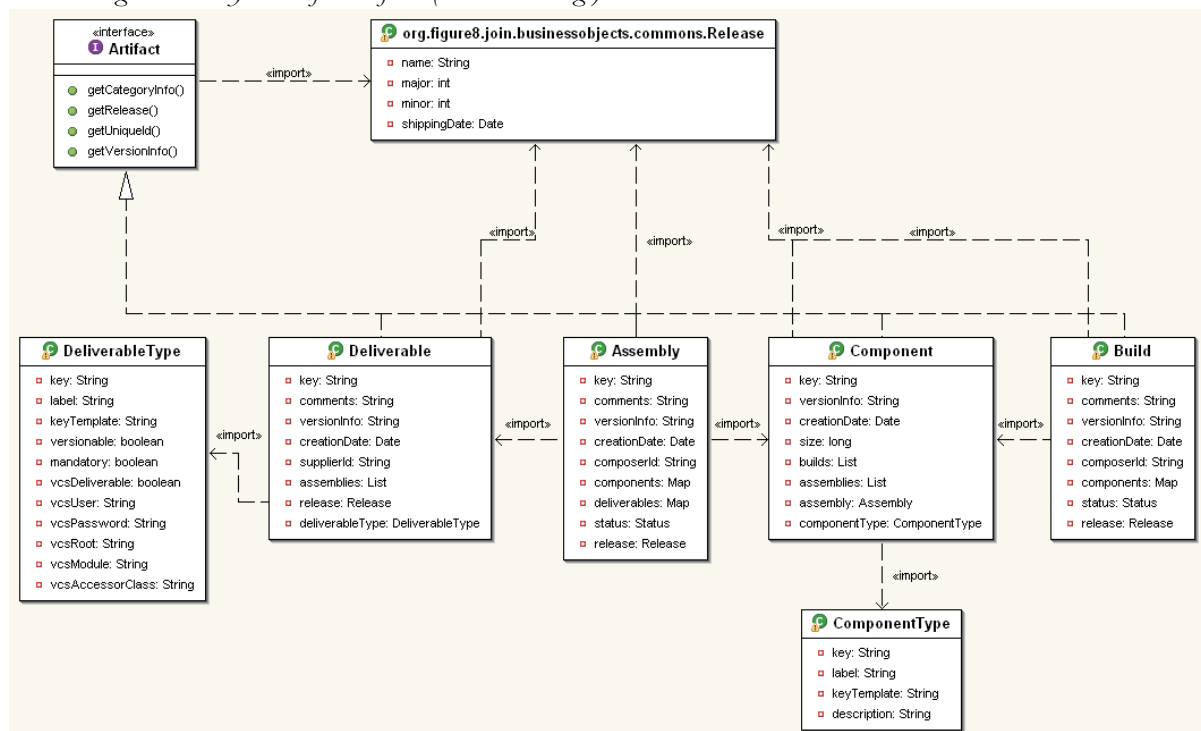
No surprise: a **Component** is an applicative unit that is deployable on its own. Components are contained into an **Assembly** - the same component may appear into many assemblies. The above definition may seem a little bit blurry regarding the exact granularity of a component... Is a library a component? Or does my whole application server represent a component? Join does not constrain you on the granularity of components to reference: it's up to you to decide which *type* of component to track; from the micro to the macro ones (you can also track both!). Thus, Join allows you to define **Component Types** in order to track components corresponding to your needs and to your project logical and technical architectures.

## Build

**Build** represents the final artifact from the integration workflow managed by Join: a production-ready package of your software project. A build is an aggregation of **Components** that have been promoted during the staging phases (through assemblies deployment onto integration environments). It is composed by picking a specific instance of component within each defined **Component Types** (some types may be optional). But why not directly using a promoted **Assembly** as a build? Most of the time this is the case, but the build composition allows you to aggregate components coming from different assemblies if you wish. Therefore you may have specialized/classified assemblies if you need it and do the final aggregation on build phase: this offers you a greater flexibility!

## Model

*Classes diagram: Entity model for artifacts (Click to enlarge)*







## 3.2 **Environments**

---

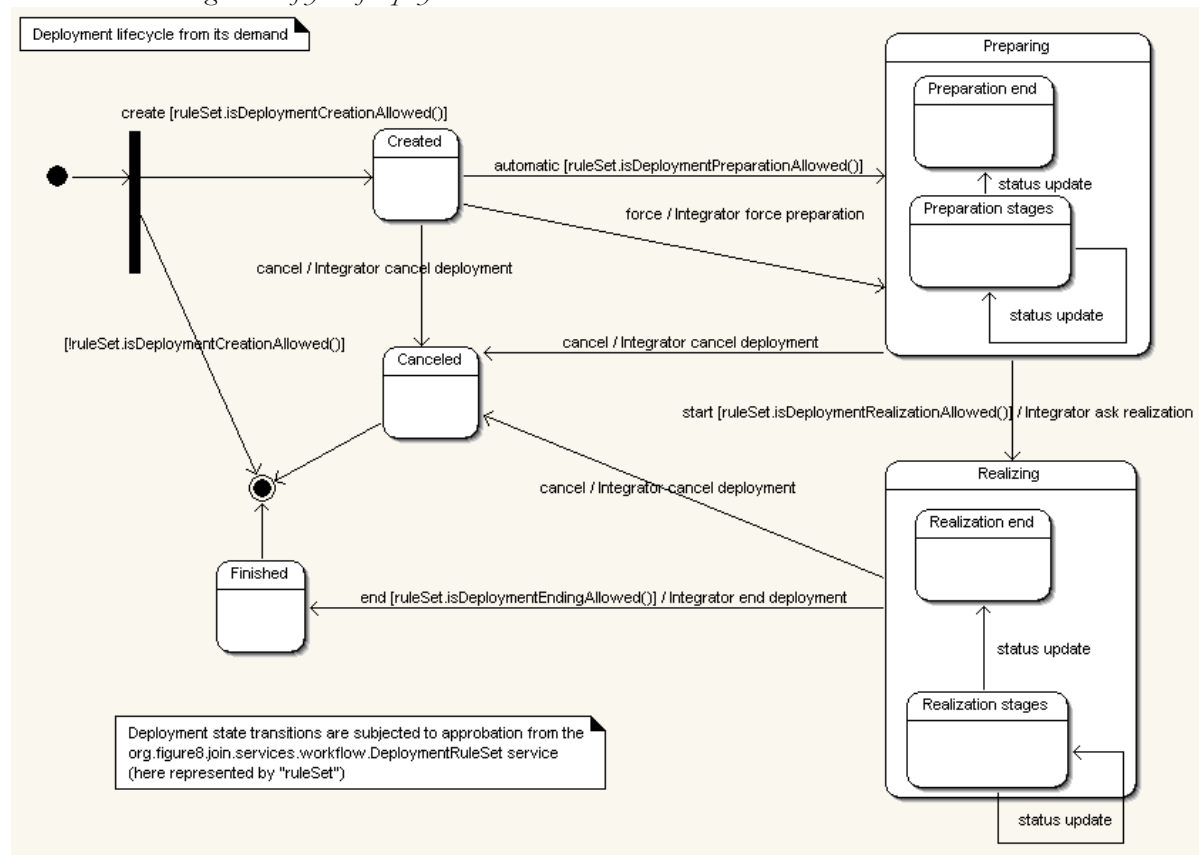
### 3.3 **Workflow**

---

### 3.3.1 Dep. Lifecycle

## Deployment Lifecycle

State-transitions diagram: Lifecycle of deployment



## 3.4 Integrator's Guide

---

## 3.4.1 Repositories

---

### Artifact Repositories

#### Configuring Repositories

This is the default repository configured into `/WEB-INF/classes/repositoryContext.xml` file:

```

<!-- Definition of repository used for storing deliverables -->
<bean id="deliverableRepository"
      class="org.figure8.join.services.repository.SimpleFileSystemRepository">
  <property name="rootDirectory">
    <value>${join.home}/deliverables</value>
  </property>
  <property name="structureDefinition">
    <value>${release}/${categoryInfo}</value>
  </property>
</bean>
<!--
  ...
  Some other repositories configuration examples
  ...
-->

```

#### Using a VFS repository

```

<!-- Definition of repository used for storing deliverables -->
<bean id="deliverableRepository"
      class="org.figure8.join.services.repository.VFSRepository">
  <property name="baseUrl">
    <!-- For a repository accessible through FTP -->
    <value>ftp://[username[:password]@]hostname[:port][absolute-path]</value>
    <!--
      For a repository accessible through HTTP
      baseUrl = http://[ username[:password]@]hostname[:port][absolute-path]
    -->
    <!--
      For a repository accessible through Webdav
      baseUrl = webdav://[username[:password]@]hostname[:port][absolute-path]
    -->
  </property>
  <property name="structureDefinition">
    <value>${release}/${categoryInfo}</value>
  </property>
</bean>

```

### **Using a JCR repository**

## 3.4.2 Domain Properties

---

### Domain Model Properties

As you have already seen it browsing the docs, many of Join runtime services may need informations on the domain model objects they are related to. As a first example there's the mailing-list notification service: we may want to use information related to original event when defining the mail body template. The same way, when you configure a new event consumer for triggering a script on a domain object creation, you will want to have sort of variables representing this object available in the execution context.

This is made possible using a special Join services called the [PropertiesExtractor](#) that is a a very simple service defining a general purpose object converter that extract a set of properties from object and its inner fields graph. But don't be afraid with that Java interface: in fact most of Join services already use this service under-cover and you'll don't have to write a single line of code...

What is really interesting here are the **properties keys produced** by the PropertiesExtractor depending on the domain model object. This document references all these properties keys. You can then use them to customize your templates, deployment parameter values or triggered scripts. You just have to wrap the properties key into an [Ant](#) style property using \$ and curly braces like this: **`${key}`**.

Here's an example of template for mail messages related to deliverable creation. `${xxx}` properties will be replaced at runtime by deliverable fields values when sending this mail. Check out [Deliverables](#) subsection for a description of properties.

```
A new deliverable of type '${deliverable.type.label}' has been supplied by
 '${deliverable.supplier.id}'.
Deliverable will be identified with '${deliverable.key}' for
 '${deliverable.release.name}' release.
Here's the supplier comments: ${deliverable.comments}.
```

### Artifacts related properties

#### Deliverables

The default implementation of PropertiesExtractor for deliverable domain objects is [DeliverablePropertiesExtractor](#). It uses a **prefix** for all produced properties that is **deliverable**. if not specified (we'll see later with assemblies that it can be overridden).

Property	Description
<prefix>key	This is the deliverable unique key (e.g. something like 'framework-r1.0-v01').

---

Property	Description
<prefix>comments	These are the comments made by deliverable supplier. This may include release note.
<prefix>supplier.id	This is the identifier of the user that has supplied deliverable (e.g. its login).
<prefix>release.name	This is the display name of the release this delivery has been made for (e.g. something like '1.0')
<prefix>type.key	This is the key of this deliverable type (e.g. something like 'fwk').
<prefix>type.label	This is the display label of this deliverable type (e.g. something like 'Framework').

## Assemblies

The default implementation of PropertiesExtractor for assembly domain objects is [AssemblyPropertiesExtractor](#). It uses a **prefix** for all produced properties that is **assembly**, if not specified.

Property	Description
<prefix>key	This is the assembly unique key (e.g. something like 'myproject-r1.0-v01-SNAPSHOT').
<prefix>comments	These are the comments made by assembly composer. This may include composition notes.
<prefix>composer.id	This is the identifier of the user that has composed assembly (e.g. its login).
<prefix>release.name	This is the display name of the release this assembly has been made for (e.g. something like '1.0')
<prefix>deliverables.<deliverable type key>.*	In addition of above properties related to assembly own informations, the default extractor add all properties related to each deliverable composing assembly. Every property of a deliverable is append to the <prefix>deliverables.<deliverable type key>. string (e.g. you may have a 'assembly.deliverables.fwk.key' property denoting the key of assembly deliverable that has the Framework type). See <a href="#">Deliverables</a> subsection for a description of available properties.

## Components

The default implementation of PropertiesExtractor for component domain objects is [ComponentPropertiesExtractor](#). It uses a **prefix** for all produced properties that is **component**, if not specified (we'll see later with builds that it can be overridden).

Property	Description
<prefix>key	This is the component unique key (e.g. something like 'component-v01').



Property	Description
<prefix>key	This is the size (in bytes) of this component.
<prefix>version	This is the information on component version (e.g. the '01' in 'component-v01').
<prefix>release.name	This is the display name of the release this component has been produced for (e.g. something like '1.0')
<prefix>type.key	This is the key of this component type (e.g. something like 'comp').
<prefix>type.label	This is the display label of this component type (e.g. something like 'Component').

## Builds

The default implementation of PropertiesExtractor for build domain objects is [BuildPropertiesExtractor](#). It uses a **prefix** for all produced properties that is **build.** if not specified.

Property	Description
<prefix>key	This is the build unique key (e.g. something like 'mybuild-r1.0-v01').
<prefix>comments	These are the comments made by build composer. This may include composition notes.
<prefix>composer.id	This is the identifier of the user that has composed build (e.g. its login).
<prefix>release.name	This is the display name of the release this build has been made for (e.g. something like '1.0')
<prefix>components.<component type key>.*	In addition of above properties related to build own informations, the default extractor add all properties related to each component composing assembly. Every property of a component is append to the <prefix>components.<component type key>. string (e.g. you may have a 'build.components.comp.key' property denoting the key of build component that has the Component type). See <a href="#">Components</a> subsection for a description of available properties.

## Environments related properties

### Resources

Property	Description
<prefix>.name	The name of bound resource
<prefix>.machine.ip	The IP address of the machine hosting this resource
<prefix>.machine.name	The name of the machine hosting this resource

## Environments

The default implementation of PropertiesExtractor for environment domain objects is [PhysicalEnvironmentPropertiesExtractor](#). It uses a **prefix** for all produced properties that is **physicalenv.** if not specified.

Property	Description
<prefix>key	This is the physical environment unique key (e.g. something like 'env01').
<prefix>name	This is the displayable name of this physical environment (e.g. something like 'Environment 01')
<prefix>resources.<resource type key>.*	In addition of above properties related to environment own informations, the default extractor add all properties related to each bound infrastructure resources. Every property of a resource is append to the <prefix>resources.<resource type key>. string (e.g. you may have a 'physicalenv.resources.webserver.name' property denoting the name of the web server that is used by environment). See <a href="#">Resources</a> subsection for a description of available properties.

## Deployments

The default implementation of PropertiesExtractor for deployment domain objects is [DeploymentPropertiesExtractor](#). It uses a **prefix** for all produced properties that is **deployment.** if not specified.

Property	Description
<prefix>id	This is the unique identifier of deployment. It has no special business meaning.
<prefix>applicant.id	The login of user that has requested the deployment.
<prefix>applicant.comments	The comments of user that has requested the deployment.
<prefix>target.name	The choosen deployment target name.
<prefix>target.description	The choosen deployment target description.
<prefix>logicalenv.key	This is the unique key identifying logical environment this deployment is for.
<prefix>logicalenv.label	This is the display label of the logical environment this deployment is for.
<prefix>physicalenv.*	In addition of above properties related to deployment own informations, the default extractor add all properties related to the physical environments deployment is made on. Every property of the environment is append to the <prefix>physicalenv. string (e.g. you will have a 'deployment.physicalenv.name' property denoting the name of the physical environment). See <a href="#">Environments</a> subsection for a description of available properties.

### 3.4.3 User Access

---

#### User Roles

Role	Description	Resource Resolver
supplier	The <i>supplier</i> role is endorsed by people providing deliverables for your software. This role is usually assigned to a development team member responsible for components releasing actions such as tagging in versioning / configuration management tool and making new component version available. In Join, this role has to be bound to a specific type of deliverables. Thus, it uses a specific implementation of <i>PermissionResourceResolver</i> .	<a href="#">org.figure8.join.services.security.DeliverableTypeResolver</a>
manager	In Join system, a <i>manager</i> is a person responsible for creating new assemblies and deciding of staging environment upgrades using these assemblies. Such a role is usually assigned to project managers for the first steps of staging process and to quality officers for later steps. For better control, this role has to be bound to a specific environment. Because environments in Join are associated to a staging process step and a software release, this allows - for example - to declare a person being manager only for the "Non regression validation of release 2.0" and not for the "End users validation of release 2.0".	<a href="#">org.figure8.join.services.security.EnvironmentResolver</a>
joiner	The <i>joiner</i> role describes a person that is one of the main users of Join : the staging or integration team member. As a member of this team, a joiner will use the Join frontend for most of its tasks : monitoring / managing artifact constructions and environment upgrades, checking environment configurations, scheduling and automating new tasks or new treatments... This role is not bound to a specific resource : it ia a global scoped security role.	None
admin		None

### 3.4.3.1 **User Definitions**

---

## 3.4.4 Scripted Process

---

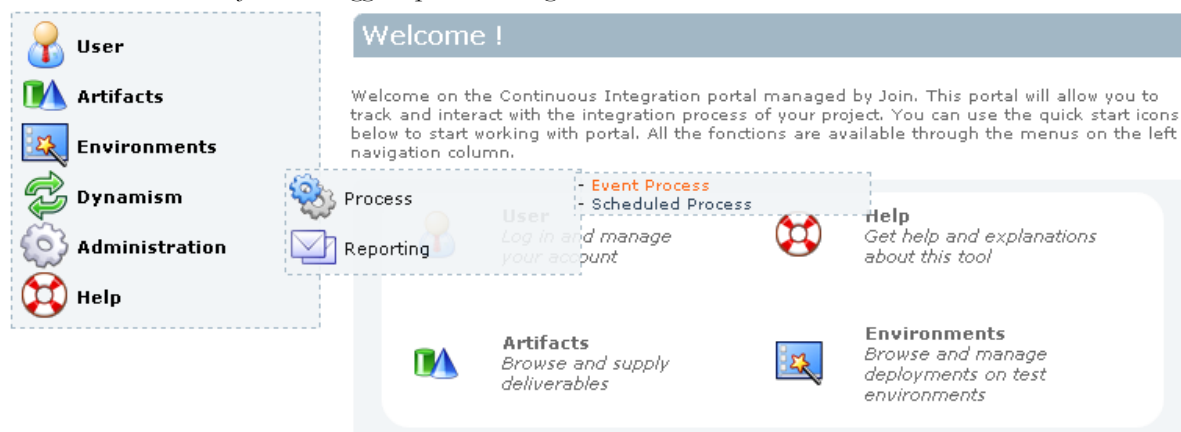
### Scripted Process Support

#### Event triggered Process

#### Scheduled Process

### Event triggered Process

*Screenshot: Menu access for event triggered process management*



*Screenshot: Event consumer edition form*

## Event consumers

The event consumers are *Java* modules allowing you to start specific process when such event appears. This process may be managed through script execution. An event consumer is defined using many elements : a **name** that has to be unique, a **queue** that allow to select the events type, a **selection expression** for fine-grained filters on events and finally, the *Java* **implementation** of this module.

### Managed event consumers

Name	Listening queue	Activation	Consumer implementation	Allows concurrent access
------	-----------------	------------	-------------------------	--------------------------

 New consumer ...

### Consumer creation

New consumer	Name :	<input type="text" value="assemblyBuilder"/>	
	Selection expression :	<input type="text"/>	
	Listening queue :	<input type="text" value="assemblyTopic"/> ▼	
	Activation :	Yes <input checked="" type="radio"/> No <input type="radio"/>	<input type="button" value="Submit"/>
	Allows concurrent access :	Yes <input type="radio"/> No <input checked="" type="radio"/>	
Consumer implementation :	<input type="text" value="Ant Script"/> ▼ Or <input type="text" value="org.figure8.join.services.scripting.ant.AntScriptL"/>		

## Scheduled Process

*Screenshot: Scheduled job edition form*


## Scheduled jobs

This page allows you to manage **scheduled jobs**. There are three job types : Ant, Groovy and Python. It is possible to schedule this jobs for just **one shot** or in a **cyclic** way.

### Managed jobs

Name  Cron expression  Type

### Edit a job

Name :   
Cron expression :    
Type :  Or

New job

User properties

Property:  Value:

## 3.4.4.1 Ant Scripts

---

### Ant Scripts Support

#### Join built-in tasks for Ant

##### Commons attributes

Attribute	Description	Required
url	The URL of the Join server you will use to realize the action represented by the custom Ant task. This URL should be the base URL of your frontend Join web application, e.g. something like <b>http://myserver.mycomp.com:8080/join</b>	Yes
user	The identifier of the user for connecting to Join remote services. This user should have the required permissions for performing the action represented by the custom Ant task. Example: user should have endorsed the <a href="#">joiner</a> security role for logging a process status	No, if <b>join.security.user</b> system property is defined
password	The credentials of the user represented by the previous task attribute.	No, if <b>join.security.password</b> system property is defined

### Process status related tasks

#### Assign a status

The first Join built-in tasks for Ant -and the more frequently used one- is the [LogStatusTask](#). You should use this task when you want to assign a specific status to the current process when you reach a defined point of your script execution. This is useful for keeping track of your process progress when important steps are done.

First, you have to define the custom **logstatus** task within a target of your script. Then, you can later use the `<logstatus .../>` task with correct attributes. Here is a typical usage example :

```
<taskdef name="logstatus"
  classname="org.figure8.join.services.scripting.ant.tasks.LogStatusTask"/>
...
<logstatus process="assembly" ref="{assembly.key}" statusKey="assemblyOK"
  url="http://localhost:8080/join" user="foo" password="bar" />
```

The LogStatusTask has the specific following attributes :



Attribute	Description	Required
ref	The reference of object to log status on. This is typically the key of the domain entity the process is related to. This may be the assembly key, the build key of the deployment identifier.	Yes
process	The name of the process the current script (or target) is dedicated to. This may be : <b>assembly, build or deployment.</b>	Yes
statusKey	The unique key identifying the status you want to assign to the entity being processed. This is one of the status you have defined when customizing your integration process.	Yes

### Assign a status on failure

The above [LogStatusTask](#) is useful when you reach a specified point into your script execution but what if you cannot reach that point because of a failure?... Failures are unpredictable by nature so we need a way to assign a status **on failure**. That is exactly the goal of the [ListenStatusTask](#), allowing you to assign a status to your process only when something goes wrong (kinda like a try/catch block !)

First, you have to define the custom **listenstatus** task within a target of your script. Then, you can later **start** the listening of failures that will assign a status before **stop** this listening on the end of your script step. Here is a typical usage example where the *assemblyKO* status will be assign to process if the directory denoted by ``${src.dir}` does not exists :

```
<taskdef name="listenstatus"
classname="org.figure8.join.services.scripting.ant.tasks.ListenStatusTask" />
...
<listenstatus action="start" process="assembly" ref="${assembly.key}"
statusKey="assemblyKO"
url="http://localhost:8080/join" user="foo" password="bar" />
  <copy todir="${dest.dir}" failonerror="true">
    <fileset dir="${src.dir}" />
  </copy/>
</listenstatus action="stop" />
```

The ListenStatusTask has the specific following attributes :

Attribute	Description	Required
action	The action to realize for build listener. This may be <b>start</b> or <b>stop</b> .	Yes

Attribute	Description	Required
ref	The reference of object to log status on. This is typically the key of the domain entity the process is related to. This may be the assembly key, the build key of the deployment identifier.	No, if <b>action</b> is set to <b>stop</b>
process	The name of the process the current script (or target) is dedicated to. This may be : <b>assembly, build</b> or <b>deployment</b> .	No, if <b>action</b> is set to <b>stop</b>
statusKey	The unique key identifying the status you want to assign to the entity being processed. This is one of the status you have defined when customizing your integration process.	No, if <b>action</b> is set to <b>stop</b>

## 3.4.4.2 **Groovy Scripts**

---

### **Groovy Scripts Support**

## 3.4.5 Remote Access

---

### Web Services Remote Access

Many of Join services are exposed as **Web Services** so that integration with other enterprise software or language is possible. You may want to use this feature in many cases:

- From **a script** - maybe launched asynchronously by Join - in order to report a process status or an environment configuration change,
- From **another software** used by your development or quality assurance teams. An example of integration use-case is an Eclipse plugin allowing developer to deliver its work into Join from the IDE (Check [here](#) for this work in progress).

Join services are currently exposed as **SOAP**, **Hessian** or **XML-RPC** web services. You will find further details for this protocols (such as access URL, interfaces and so on) on specific pages:

Protocol	Documentation
SOAP	SOAP is a W3C recommended protocol for message exchange using XML over various transport layers. Specifications can be found <a href="#">here</a> . Specific documentation on Join implementation is located <a href="#">here</a> .
Hessian	Hessian is a binary web services protocol from <a href="#">Caucho</a> . Documentation for Join Hessian services is <a href="#">here</a>
XML-RPC	Simple Remote Procedure Call protocol in XML. Many implementations are referenced on <a href="#">Xml-Rpc site</a> . Specific documentation for Join can be found <a href="#">here</a> .

## 3.4.5.1 SOAP

---

### SOAP Web Services

SOAP is a W3C recommended protocol for message exchange using XML over various transport layers. Specifications can be found [here](#). The Join implementation of SOAP web services only used the HTTP transport layer. It uses the **XFire** stack to expose its APIs through SOAP.

#### Services

[ArtifactService](#)

[ParameterService](#)

[ProcessControlService](#)

[ResourceService](#)

#### Parameters

These are some parameters and details on how to call SOAP web services:

- The URL for SOAP requests is  
`http://host[:port]/[context]/remoting/xfire/[service]`,
- The URL for service WSDL is  
`http://host[:port]/[context]/remoting/xfire/[service]?WSDL`,
- Parameters and returned objects APIs can be found in WSDL or from [JavaDoc](#).

## 3.4.5.2 Hessian

---

### Hessian Web Services

Hessian is a binary web services protocol from [Caucho](#). It is lightweight, efficient and easy to use.

#### Services

[ArtifactService](#)

[ParameterService](#)

[ProcessControlService](#)

[ResourceService](#)

#### Parameters

These are some parameters and details on how to call Hessian web services:

- The URL for Hessian requests is  
`http://host[:port]/[context]/remoting/hessian/[service]`,
- Parameters and returned objects APIs can be found from [JavaDoc](#).

### 3.4.5.3 XML-RPC

---

#### XML-RPC WebServices

Xml-Rpc is a simple Remote Procedure Call protocol in XML. Many implementations are referenced at [Xml-Rpc home](#). Join uses the [Apache Xml-Rpc](#) implementation for Java on the server side to expose its APIs.

#### Services

All available services are exposed via the `XmlRpcHandler` interface. You can find API documentaion [here](#).

#### Parameters

These are some parameters and details on how to call Xml-Rpc services:

- The URL for Xml-Rpc requests is `http://host[:port]/[context]/xmlrpc`,
- All services name should be prefixed by `join..` For calling the `setBuildStatus` service, the method name is `join.setBuildStatus`,
- All keys in struct are case sensitive,
- All strings are passed as UTF-8, and not ASCII per the XML-RPC update on 6/30/2003.

#### Samples

This is a sample on how to call a Join service from a [Groovy](#) script. Because Join has built-in support for launching Groovy scripts, this is a useful piece of code for making your script report a process status...

```
...
import groovy.net.xmlrpc.*
def service = new XMLRPCServerProxy("http://localhost:8080/join/xmlrpc")
def token = service.join.login("mylogin", "mypassword")
println "Got a security token: ${token}"
service.join.setBuildStatus(token, "build_ok", buildId)
...
```

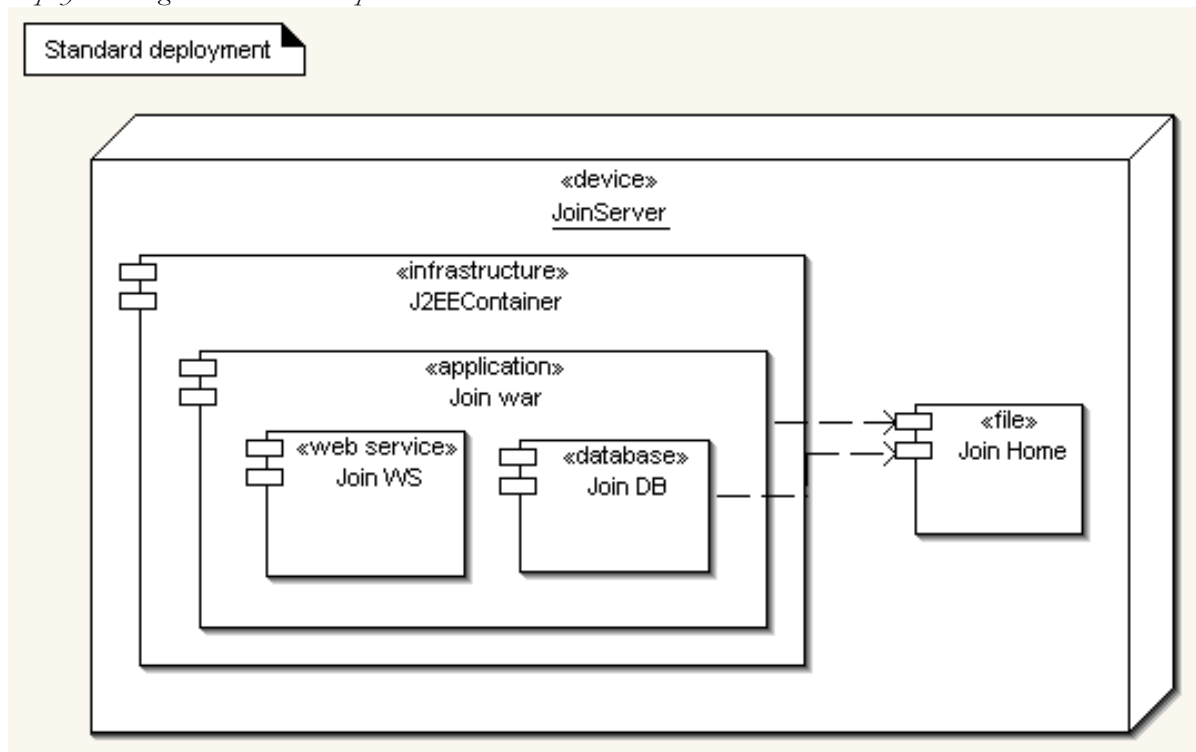
Another sample in pure Java can also be found into Join [tests suite](#).

## 3.4.6 Setup Modes

---

### Standard Setup

*Deployment diagram: Standard setup mode*

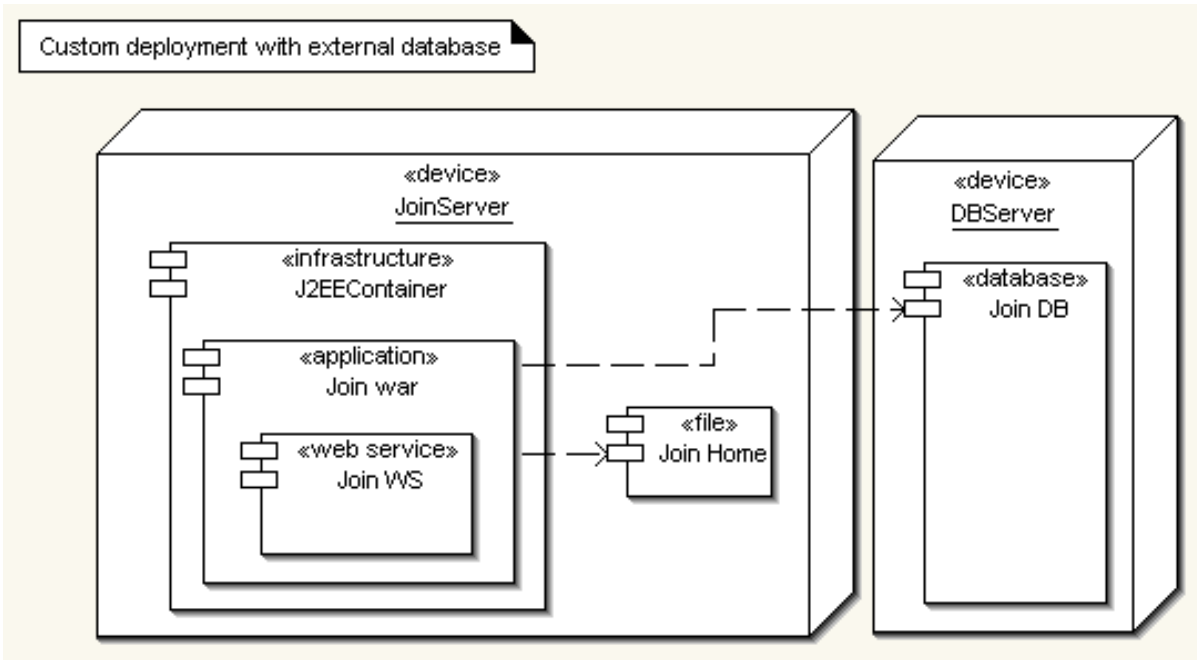


### Custom Setup

#### Custom Setup with external database

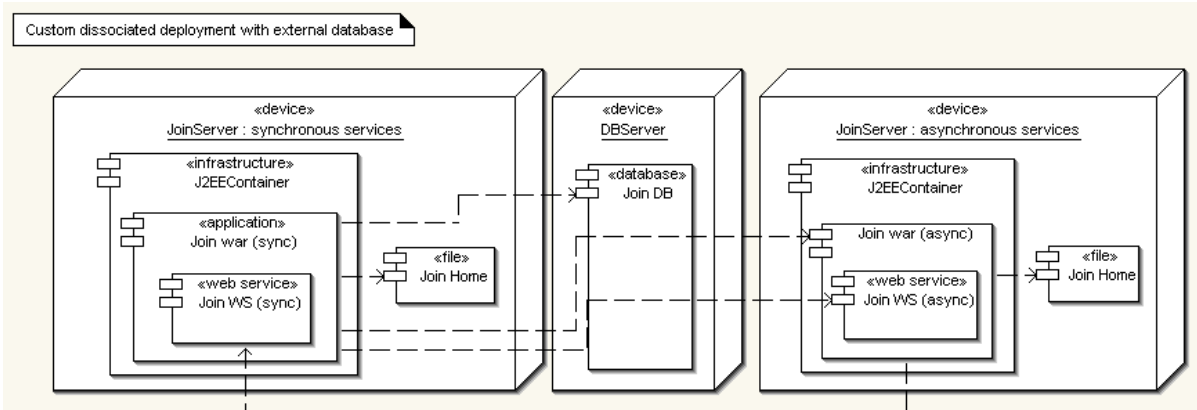
*Deployment diagram: Custom setup with external database*





**Custom Dissociated Setup with external database**

*Deployment diagram: Custom dissociated setup with external database*



## 4.1 **Ant**

---

<http://ant.apache.org>

## 4.2 **Spring**

---

<http://www.springframework.org>

## 4.3 **ActiveMQ**

---

<http://activemq.apache.org>