calendar
bedework
www.bedework.org

Bedework Calendar Deployment Manual

Bedework version 3.0

Last modified: May 8, 2006

Bedework Deployment Manual

The Bedework Deployment Manual contains instructions for customizing and installing a production version of Bedework. It begins with setting up your build environment and ends with guidance on creating your initial calendars, subscriptions, views, and admin users. For instructions on customizing the look and layout of your production system, please see the Bedework Design Guide on the Bedework website: www.bedework.org.

Contents

Prerequisites

Requirements

- JDK 1.4
- JAVA_HOME environment variable must be set
- To implement the Bedework calendaring system, it is useful to understand the following:
    - Java servlets: http://java.sun.com/products/servlet/docs.html
    - Servlet containers (e.g. Tomcat, JBoss)
    - Authentication is local to your site - some Java programming may be necessary to accomplish this

Install the quickstart and test your environment

Before attempting any customization, please test your environment by running the quickstart release. It is always wise to test your changes incrementally; test each small change to make certain you understand its effects. Doing these two things will help you understand the system and will provide useful information to the Bedework support community if you run into trouble and wish to ask for help.

Prepare a localized version of Bedework

Prepare your build environment

Bedework uses ant for the build and deploy process and a number of property files are used to control that process. Ant properties have the characteristics that once set they cannot be modified so to override default settings you need to set them earlier in the process.

The build first looks for a property file called **bedework.build.properties** in your home directory. Here you can set properties which will override the default settings. In particular you can tell the build system the name of the configuration you want to build with the setting (e.g):

```
# Name prefix for the property file in config/configs/
```

```
org.bedework.clone.default=myconfig
```

This would cause the build to include the file config/configs/myconfig.properties.

The default value for this property is democal which means the system builds and deploys using the file config/configs/democal.properties. **Do not change the original file.** Please keep it for reference.

Make a copy named appropriately and set the org.bedework.clone.default to use it.

Copy and update the skins (templates)

Now is the time to localize the skins. This can be as simple as replacing the title graphics and text, or as involved as modifying the global layout and behavior of the front-end. Please see the Bedework Design Guide for instructions on updating layout and styles.

Stylesheets and Path discovery

Your XSL stylesheets and associated template images and resources (e.g. css files) are located on a web server at the url specified by "approot" in the config file (e.g. config/configs/myconfig.properties). Each of the three web clients (public/guest, personal, and admin) has an associated approot where these files are to be found by the system. For example:

```
org.bedework.webadmin.app.root=http://somewebserver/bedework-3-0-admin
:
org.bedework.webpubevents.app.root=http://somewebserver/bedework-3-0-
guest
:
org.bedework.webpersonal.app.root=http://somewebserver/bedework-3-0-
personal
```

Placing the stylesheets and resources on a separate web server (we suggest/encourage you to use your primary web server) will make them significantly more convenient to access and manipulate. Note that if you choose to serve a client over https, you may wish to specify the same protocol for the approot to avoid mixed content messages. Note that the stylesheets are *only* loaded at first reference (or if an administrator explicitly flushes the stylesheets).

Under this is a 3 tier structure based on:

- locale
- user agent
- stylesheet name.

The top two are normally named "default". The filters will work down the structure trying a specific name first then trying default. For example, in the locale "fr_CA" a path

```
{$appRoot}/fr_CA/default/...
```

could lead to a set of French stylesheets while

```
{$appRoot}/default/default/...
```

could lead to English.

The discovered 'real' path is cached with the 'idealised' path as the key so that subsequent lookups for the same path will proceed without the discovery phase.

A deploy time property can specify that the target server has directory browsing disallowed, in which case the filters search for a marker file called xsltdir.properties

Set up a production database

The quickstart is set up to use hsqldb and this provides a good way to try out the system. To move to another database system you will need to configure the build to affect the following:

- Configure tomcat to use another database
- Configure hibernate to use the appropriate dialect
- Configure the schema process to point to the same database
- Rebuild
- Configure the dump/restore

Configure Tomcat 5.0.28

Tomcat 5.0.28 is configured by editing conf/server.xml. Find the resource defined something like the following:

```
<DefaultContext>
   <Resource name="jdbc/calDB" auth="Container"
             type="javax.sql.DataSource"/>
   <ResourceParams name="jdbc/calDB">
     <parameter>
       <name>username</name>
       <value>sa</value>
     </parameter>
     <parameter>
       <name>password</name>
       <value></value>
     </parameter>
     <parameter>
       <name>driverClassName</name>
       <value>org.hsqldb.jdbcDriver</value>
     </parameter>
     <parameter>
       <name>url</name>
       <value>jdbc:hsqldb:hsql://localhost:8887</value>
     </parameter>
     <parameter>
       <name>maxActive</name>
       <value>8</value>
     </parameter>
     <parameter>
       <name>maxIdle</name>
       <value>4</value>
     </parameter>
   </ResourceParams>
 </DefaultContext>
```

You will need to change at least the name of the driver class, the url and possibly the username and password. You will also need to ensure that the driver jar is added to the common/lib directory.

Configure hibernate

Setting your SQL dialect

Configuring hibernate to use the appropriate dialect is done in a few places. For the application build you need to set the dialect in your configuration file in config/configs.

For example, if your **bedework.build.properties** specifies a clone name of **myconfig** (see Setting up your local configuration) then in the file **config/configs/myconfig.properties** you need to set the property

```
org.bedework.global.hibernate.dialect={yourDialectHere}
```

where {yourDialectHere} is an accepted hibernate SQL dialect such as org.hibernate.dialect.HSQLDialect or org.hibernate.dialect.MySQL5Dialect.

A list of dialects understood by Hibernate can be found at:

http://www.hibernate.org/hib_docs/v3/reference/en/html/session-configuration.html#configuration-optional-dialects

For the most up-to-date listing, look at the actual org.hibernate.dialect classes in the Hibernate source (for example, to use MySQL 5 which is not currently listed in the on-line Hibernate documentation, use org.hibernate.dialect.MySQL5Dialect).

Building the Schema

For schema builds you need to provide a separate **hibernate.properties** file which is configured to use jdbc directly. For example, if user douglm creates a file **.bedework/hibernate.properties** in his home directory then the property

```
org.bedework.schema.hibernate.properties=/home/douglm/.bedework/hibernate.properties
```

in his **bedework.build.properties** files will direct the schema and dump-restore to use that file. The file should look something like:

```
hibernate.query.substitutions true 'T', false 'F', yes 'Y', no 'N'

hibernate.dialect org.hibernate.dialect.HSQLDialect
hibernate.connection.driver_class org.hsqldb.jdbcDriver
hibernate.connection.username sa
hibernate.connection.password
```

```
hibernate.connection.url jdbc:hsqldb:hsql://localhost:9001

#################################
### Hibernate Connection Pool ###
#################################

hibernate.connection.pool_size 1

###############################
### Proxool Connection Pool###
###############################

## Properties for external configuration of Proxool

hibernate.proxool.pool_alias pool1

## print all generated SQL to the console

hibernate.show_sql false
## set the maximum JDBC 2 batch size (a nonzero value enables
batching)

hibernate.jdbc.batch_size 0

## use streams when writing binary types to / from JDBC

hibernate.jdbc.use_streams_for_binary true


## set the maximum depth of the outer join fetch tree

hibernate.max_fetch_depth 1

############################
### Second-level Cache ###
############################

## optimize chache for minimal "puts" instead of minimal "gets" (good
```

```
for clustered cache)

#hibernate.cache.use_minimal_puts true

## enable the query cache

hibernate.cache.use_query_cache true

## choose a cache implementation

hibernate.cache.provider_class
net.sf.hibernate.cache.HashtableCacheProvider
```

Configure dump/restore

The dump/restore target uses yet another property file from your home directory **bedework.dumprestore.properties**. This file allows you to overide the default settings for the dump/restore which are found the configuration. The restore process uses xml data generated by the bedework dump process. Your **bedework.dumprestore.properties** will look something like:

```
org.bedework.dump.arg.dumpfile=${user.home}/.bedework/bwcaldata.xml

# -debug or -ndebug
org.bedework.dump.arg.debug=-debug

# hibernate db type
org.bedework.dump.arg.hibernate.dialect=org.hibernate.dialect.HSQLDia
lect

# Hibernate jdbc parameters
org.bedework.restore.arg.jdbcdriver=org.hsqldb.jdbcDriver
org.bedework.restore.arg.jdbcurl=jdbc:hsqldb:hsql://localhost:8887
org.bedework.restore.arg.jdbcid=sa
org.bedework.restore.arg.jdbcpw=
```

```
# This is where we get the data.
org.bedework.restore.arg.dumpfile=/home/douglm/calendar3/tools/resour
ces/initbedework.xml

# Used to fix up orphaned events.
org.bedework.restore.arg.defaultpubliccal=-defaultpubliccal
"/public/Other Events/Other"
```

The dump/restore process builds a hibernate configuration using the above parameters.

Rebuild and redeploy the system

Issue the following commands within the quickstart directory:

```
ant clean.deploy.debug
```

or

```
ant clean.deploy
```

Build schema and initialize

Before building the schema, if you are building for a database other than the quickstart hsqldb, you will need to make the appropriate jdbc drivers available. Place the driver jar file in the directory calendar3/lib/jdbc and it will appear on the class path for the schema and the dump and restore targets.

With all that in place it is now possible to create a schema and initialize the database. Create a schema for your system with:

```
ant schema
```

and reply no to the "text only?" message. The hibernate schema tool will create all the tables and constraints. If you reply 'yes' it will just create a sql file in your calendar base directory.

To populate the database use

```
ant restoredb
```

and reply **y** to the prompts and dire warnings.

Authentication

The calendar uses container-based authentication as defined by the Java servlet specification. There is no authentication code within the calendar system.

Authentication can be managed by the servlet container in a number of ways which are currently beyond the scope of this document. Tomcat's implementation can be configured, at its simplest, in its <tomcat>/conf/tomcat-users.xml file. The tomcat website provides details on configuring tomcat to use other forms of authentication, including ldap and databases. (see also Tomcat SSL HowTo).

An alternative which has been implemented is to use filter based Yale/JA-SIG CAS.

Build and deploy

1. Build the calendar with the command:
   `ant clean.deploy.debug`
   (".debug" will provide debugging output in the server log. You can remove it and rebuild when you are convinced you are production ready.)

   This will create a number of WAR files in <tomcat>/webapps/ including:
   cal.war, caladmin.war, and ucal.war

2. If you are using the Tomcat directory within the quickstart distribution, your application war files are now deployed. Otherwise, collect the war files and drop them in your container.

Create initial calendars, subscriptions, and views

1. Within the /caladmin application, select: "Manage calendars" from the Main Menu. You will see a default set of calendars that you can manipulate.

2. When you are satisfied with your collection of calendars, select "Manage subscriptions" from the Main Menu. You will see a default set of subscriptions that map to top level calendars. A one-to-one relationship does not have to exist between top-level calendars and subscriptions, but it is useful for initially setting up your views. Set any subscription preferences here, including css style.

3. Finally, create views by selecting "Manage views" from the Main Menu. Create views

by adding subscriptions to the view. Again, a one-to-one relationship does not have to exist between top-level calendars and views, but in many cases, this is what you'll want to expose in your public calendar system. The views named here will present themselves in the pull down list of views in the public client.

By default, there is a view named “All” that contains all existing subscriptions. The “All” view is set as the default view for the special user “public-user”. If you want to change the default view, enter the special user “public-user” in the “Edit user preferences (enter userid):” field in the User Management section of the Main Menu; you will be presented with a form in which you can change the default view.

Add groups and users

1. Within the /caladmin application, select "Admin Groups: Add"
2. Give the group a name, a description, provide a userId for the owner, and set the "Events Owner" to agrp_*groupName*, or something meaningful that will will easily identify which group the event belongs to. (Note: you should leave the prefix on group owners; the prefix is defined on line 112 of calendar3/appsuite/clones/myorg.properties.)
3. Once the group is added, add members by providing a userId that should map to the userId used to authenticate to the admin client.