calendar
bedework
www.bedework.org

Bedework Calendar Deployment Manual

Bedework version 3.1

Last modified: July 30, 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 administrative 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 and runtime properties

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 location of the configuration you want to build with the setting (e.g):

```
# Location of our bedework property files
```

```
org.bedework.config.properties=${user.home}/bwbuild/myconfig.properties
org.bedework.config.options=${user.home}/bwbuild/myconfig.options.xml
```

This would cause the build to include the files

<home>/bwbuild/myconfig.properties and

<home>/bwbuild/myconfig.options.xml

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

Make copies named appropriately and set the properties above to use them.

The properties file is mostly associated with the deployment process while the options.xml file is for runtime properties. Gradually this demarcation is being cleaned up so that the properties file will eventually not be included on the class path. Some properties are needed at deployment and at run time. These are copied during deployment from the properties file into the options file.

The properties file.

The properties file is divided into sections with different property prefixes.

Install

The section prefixed "org.bedework.install" defines which applications are to be installed. This consists of a list of application names.

For each name there should be a corresponding section prefixed with "org.bedework.app.<name>" and also a corresponding section in the options file.

Global

The section prefixed "org.bedework.global" defines properties global to the whole deployment process.

Application

The section with properties prefixed "org.bedework.app.<name>" are the application deployment properties, one section per named application.

Two properties define the project and type of application. The value of the property "org.bedework.app.<name>.project" defines which project the application is a part of. Currently these can be "caldav" or "calendar3". (There is also a "freebusy" project).

The value of the property "org.bedework.app.<name>.type" corresponds to the name of a subdirectories in <project>/deployment, e.g. webpublic, webadmin, etc. So to define the administrative client named CalAdmin of type webadmin we have the fragments:

```
org.bedework.install.app.names=...,CalAdmin,...
...
org.bedework.app.CalAdmin.project=calendar3
org.bedework.app.CalAdmin.type=webadmin
```

Multiple versions of each application type may be deployed, each configured differently. This is of importance for calendar suites (departmental calendars).

The options file.

This xml file contains run time properties and is divided into sections much like the properties file. Some values may be copied out of the properties file if they affect both the deployment and run time. Most of the options are used to set field values in named classes so that the application will load the settings once only with a single call.

It is important to set the system properties for a new system. These are found in the "syspars" section of the properties file. A number can be left with the default values and some are not yet implemented. The properties it is particularly important to set (and their default settings are:

```
<name>bedework</name>
<tzid>America/New_York</tzid>
<systemid>demobedework@mysite.edu</systemid>
```

The name property appears in the system table as the key. The tzid is the default timezone to be used for times and dates The systemid is used when generating guids for calendar entities. This name should be related to your site for ease of identification and if you run multiple systems should be different for each.

There are also a number of names used when creating default calendars. These should be set to some appropriate localized value.

The size settings are mostly unused at the moment.

The property

```
<userauthClass>org.bedework.calcore.hibernate.UserAuthUWDbImpl
</userauthClass>
```

is used by the administrative client when checking access to the client. The group class setting are explained in the "Access rights and groups" section below.

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 the property "app.<name>.root" (and "app.<name>.cal.suite" for the public client) 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, given the values:

```
org.bedework.app.CalAdmin.root=http://somewebserver/bedework-3-1-admin
org.bedework.app.Events.root=http://somewebserver/bedework-3-1-guest
org.bedework.app.Events.cal.suite=MainCampus
org.bedework.app.SoeDept.root=http://somewebserver/bedework-3-1-guest
org.bedework.app.SoeDept.cal.suite=SoeDept
org.bedework.app.UserCal.root=http://somewebserver/bedework-3-1-personal
```

the administrative client and user client stylesheets will be found at the given url while the Events public (guest) client will have its stylesheets located at
http://somewebserver/bedework-3-1-guest.MainCampus
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 will need 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>
           <parameter>
             <name>defaultAutoCommit</name>
             <value>false</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. The defaultAutoCommit setting is particularly important and also easy to miss. Without that setting it defaults to true which effectively disables transactions

and prevents rollbacks. This can lead to inconsistent databases.

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 your configuration file.

For example, if your **bedework.build.properties** specifies a properties file at **<home>/bwbuild/myconfig.properties** (see Setting up your local configuration) then in that file 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).

Before building the system, 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 applications.

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

With all that in place it is now possible to create a schema and initialize the database. The deploy process created a zip file in the calendar3 dist directory which can be unwrapped to run the schema build. The default name for that application is dumpres.zip. This file contains a Unix shell script named bwrun.sh and a Windows bat file named bwrun.bat.

Unzip that file to a convenient location and in linux ensure the file called bwrun.sh is executable (chmod +x bwrun.sh).

To create a schema file for your system enter:

```
./bwrun schema
```

The hibernate schema tool can also create all the tables and constraints. If you wish to use this option then enter

```
./bwrun schema-export
```

A word of warning here; while the schema-export option is useful there is little or no error checking taking place. Check back through the output for errors.

To populate the database use

```
./bwrun restoredb <filename>
```

where <filename> is the name of a dump file produced by the dump process.

To initialize an empty database use initdb, e.g.

```
./bwrun initdb
```

This uses an initial data file wrapped up with the dump restore application.

The application can be used to produce a regular nightly xml dump of the data. In future releases the dump/restore will be reformatted to facilitate restoration of a single users data.

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.

Add administrative groups and users

In the quickstart administrative groups are stored in the calendar database. Administrative groups are intended to be separate from user groups to allow different access rights to be defined for administrative users. (See "Access rights and groups" below)

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 in the properties file as the property org.bedework.app.CalAdmin.admingroupsidprefix)

3. Once the group is added, add members by providing a userId that should map to the userId used to authenticate to the administrative client.

Add calendar suites

Note: When the database is initialized, you begin with a clean slate. You cannot run the public client until a calendar suite has been defined in the admin client. These steps document setting up an example calendar suite; note that the name for the suite must match the org.bedework.app.Events.cal.suite property in your configuration properties file.

1. Within the /caladmin application, select "Manage Calendar Suites"
2. There should be at least one for the main campus calendar called “MainCampus”.
3. If it does not exist create it and make the group owner be “campusAdminGroups” and the root calendar be “/public"

Create initial public 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.

Access rights and groups

Access to resources in Bedework is controlled by an access control system based on WebDAV and CalDAV. A user's access is based on their identity and group membership. The system allows a different group structure for administrative and user access. This ensures that a given user, when logged in to the user client, does not have any special administrative rights which may lead to unfortunate consequences, such as deleting a public calendar by mistake, or adding private events to public calendars.

Two system parameters determine this behavior, the initial values are set in the xml options file, democal.options.xml in the quickstart. The relevant elements in the syspars section are:

```
<admingroupsClass>org.bedework.calcore.hibernate.AdminGroupsDbImpl
</admingroupsClass>
<!--
<usergroupsClass>org.bedework.calcore.ldap.UserGroupsLdapImpl
</usergroupsClass>
 -->
 <usergroupsClass>org.bedework.calcore.hibernate.GroupsDbImpl
 </usergroupsClass>
```

Note the second of the three is commented out. These settings define which class will be used to manage groups for the administrative and user clients. The first class is an implementation which uses a the Bedework database to store the administrative groups.

The last setting is a dummy class which does nothing but which could use the database to allow testing or perhaps even for sites which don't want to use a site-wide directory.

The commented out setting is a class which uses ldap to determine group membership. In the options file is a section labeled "user-ldap-group" which configures this class. This class should also be usable with Active Directory.

Personal calendars, subscriptions and views.

The default state for a personal calendar user is to have one view with a name determined by the "defaultUserViewName" syspars setting (default "All"). This view contains one subscription to the user root collection at "/user/<account>" with the user account as the name. Only the default calendar with the name given by the "userDefaultCalendar" syspar is created.

Other special calendars, such as Trash, Deleted, Inbox etc are only created as needed.

The initial default setting is normally created at the first service api open for that user. Because all (non-special) calendars in a subscription are visible, if a user creates a new calendar it will automatically be visible in the default view. Thus the initial default state is relatively simple for users to manage and will probably be sufficient for most users.

When a user explicitly subscribes to a calendar, such as a public calendar or one shared by another user, that subscription will be automatically added to the default view. Once again, this default behavior is suitable for most users.