calendar bedework

www.bedework.org

Bedework Calendar Deployment Manual

Bedework version 3.3

Last modified: January 24, 2007

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.

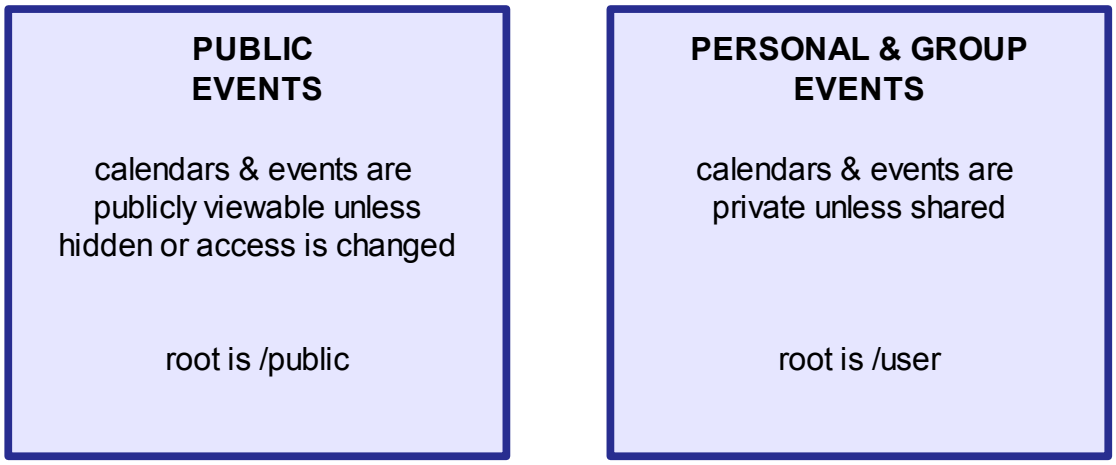
Table of Contents

1. Overview

Calendar collections and folders.

All calendar entities, events, todos etc are stored within calendar structures consisting of 'calendar collections' and 'folders'. The distinction between folders and calendar collections is a requirement of CalDAV, one of the calendaring systems supported by bedework. A folder may not directly contain calendar objects such as events. It may contain calendar collections or other folders. A calendar collection may only contain calendar objects and may not contain folders.

The Bedework system is divided into two main spaces: the public events space, and the personal and group calendaring space. Public events are stored below a public calendar root folder and personal calendars are below the user calendar root folder.



There are a number of types of calendar collection:

1. **Collection**: the usual kind of collection of events, todos etc.
2. **Trash**: when events are 'deleted' they are moved here if possible. This gives users the opportunity to undo the delete by moving it back.
3. **Deleted**: this is used when the user does not have delete access to an entity. For example, a user might be subscribed to films and want to delete some of them from the calendar view. An annotation is created and placed in this special calendar and acts as a mask to suppress the entity. This may be changed in the future for some filter based approach.
4. **Busy**: this type of calendar will be used to hold busy time information which will be included in a free/busy display. For example, if the user always has lunch at 12midday

to 1pm, then a recurring object could be placed here so that the calendar view is not cluttered by such entries.

5. **Inbox**: this is where incoming meeting requests or published event information will appear.

6. **Outbox**: scheduling requests targetted at users on other systems will be placed here for subsequent processing.

Subscriptions

Subscriptions are made to calendars. At the moment bedework only supports subscriptions to calendars within the same system. In time we hope to allow subscriptions to external calendar systems. A subscription must have a unique name for the owner and may reference either a folder or a calendar collection. A calendar must have a subscription before it can be made visible in the personal user or public events client.

By default a user has a single subscription to their 'home' folder, e.g. a user 'janet' would have a subscription to "/user/janet". There are no default subscriptions in the public space. They must all be explicitly created by administrators. Associated with each subscription is some styling information which allows for coloring or other stylistic preferences.

Views

Views are named collections of subscriptions. They are of most use in the public events system though they are used in personal calendars. Public events administrators can create a number of views. Typically they are used to group together subscriptions are to hold all subscriptions (as in an "All" view).

Public Events Calendaring

The point of a public events system is, in general, to disseminate event information to the widest and most appropriate audience possible. It is important that the public events are easily navigable and straightforward to find, subscribe to, and create. To achieve this, the calendar tree should be topical, consisting of high-level aggregate categories such as "Arts", "Lectures", and "Athletics".

Illustration 1 shows an example calendar structure for public events. Noticeably absent from the tree are folders named by

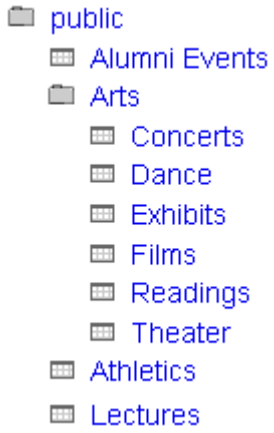


Illustration 1: public events

organization or department. As we have learned in web information architecture, users should not need to know what group within an organization's hierarchy sponsors an item to find it. However, an organization should still be able to pull a "departmental" view of the calendar information and present only that to their intended audience: this is the purpose of Calendar Suites. These are used to display a customized and filtered view of public events, but are not intended for group calendaring – which takes place within the Personal & Group calendaring space.

We believe the most flexible architecture for delivering public events is to create a large central pool of events organized by topic and to draw from this pool both for individual user and departmental access. Users who are interested in all lectures across your instituion will be able to subscribe to a single source for these events, while departments may display only the lectures sponsored by them from the central pool.

As stated above Bedework's calendar trees are divided into folders and calendar collections. Only calendar collections may be added to folders. Events may only be added to calendar collections. In Illustration 1, it would not be possible to add an event to the root "Arts" folder; rather, an administrator must decide to which art calendar an event belongs. To keep this decision making process from becoming onerous, calendar collections within the top-level categories should be course-grained, keeping the selection of appropriate calendars for events as simple as possible.

Public Event Administration: Users & Groups

Groups:

In Bedework's current system, public events are administered centrally through the administrative web client. Every administrator is a member of a group, and events are owned not by administrative users directly, but by a special "event owner" associated with the group. Each group also has a group owner that is a specific administrative user; this role does not currently provide any functionality, but may in time (such as managing group membership).

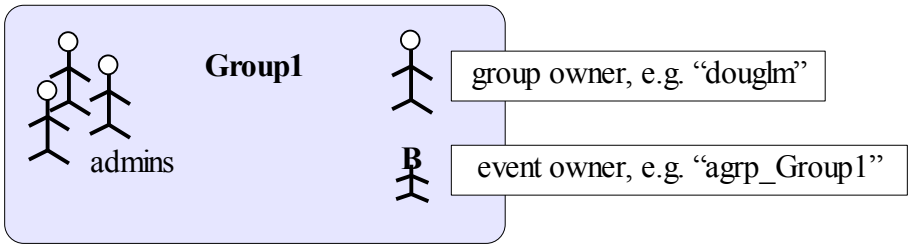


Illustration 2: Group Structure

The event owner is a user, not found in the organizations user space, that owns all the events for a group. Having such an owner allows all group members to see and edit events within the group. Also, the event owner is the "user" who's preferences define the behavior of a calendar suite. The system ensures that these event owners are distinct from real users by prefixing them with a string, by default "agrp_".

Within the administrative client events are filtered by the current event owner so that administrators can only see and edit events for their current group.

The super user(s) can switch to any group making all events available.

In addition, if the system is appropriately configured, locations, contacts and categories are also filtered making them editable only by the group. They are however, always readable. The other extreme of having no editable locations, contacts and categories has been requested and something near that can be achieved by creating them in a special group. The intent is to create locations so that they conform to a particular code of practice. Not allowing administrators to create locations at all may become a problem when events take place off campus.

Group structure and access control:

Access control is inherited from the top down the group tree. Therefore, it is best to create a single, top-level group for access to /public and then add all other admin groups to it. By default, Bedework comes with a top-level group named "campusAdminGroups". All other groups should be made members of this group to inherit write-content access on /public. Unlike calendars, groups are organizational, e.g. "Arts", "SOE" (school of engineering), or "Athletics".

While it is possible to close branches of the /public sub-tree from access to all administrators by creating a different group hierarchy, we discourage doing this. If public calendars are kept topical, an administrator from any group can add events to any calendar in the tree. However, an administrator can only see events created by his or her group.

It is important to remember that "departmental" calendaring (group calendaring) should be kept away from the /public tree – which is only intended for public events. Events that must not be seen by any but a subset of your community are not public (such events are to be distinguished from those intended for a subset of your community that are ok for others to see). These should be kept in the personal space, or if your need dictates it, in a top-level /dept branch of the tree which you will need to create. We believe in actual practice, most group calendaring will best be managed within the personal and group space.

Administrative users:

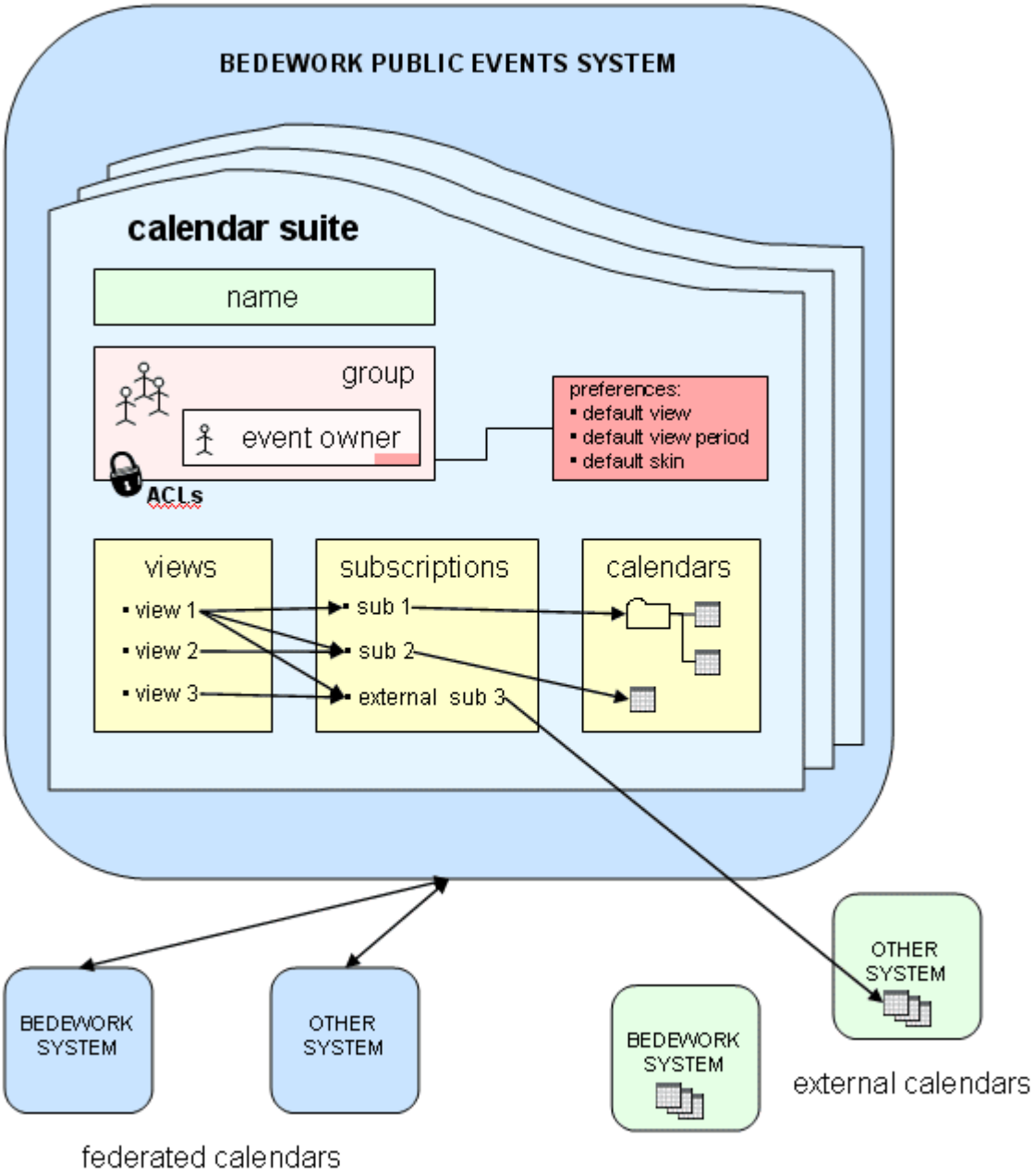
To add an administrative user to the system, simply add the user to a group. When a user is removed from all groups, the user will be removed from the system. Because the public event space is distinct from the personal calendaring space, administrative users are managed in Bedework's database by default (though they need not be).

Once in the system, an administrative user may be given roles from the “Manage public event administrators”menu item under “User management”. The current meaningful roles are 'super-user' and 'publicEvent' administrator. The publicEvent role allows write-content access to the /public tree (assuming the user is in a child group of the top-level campusAdminGroups and the ACLs are not changed); the super-user role allows full access to the Bedework admin client.

Authentication

Authentication is not managed by the Bedework system internally, but by the servlet (or J2ee) container (see section 2.4, Authentication). Administrative roles and authentication are separate; a user who can authenticate, but who is not an administrative user, will see a “No Access” message upon logging into the admin client. A user who has been added as an administrator but who cannot authenticate cannot log in at all. Both conditions – authentication and authorization – must be met for administrators to work in the admin client.

Calendar Suites



The intent of calendar suites is to provide a customized view and appearance for significant groups within an organization, such as a school or large department. The calendar suite is associated with an administrative group which has an event owner (an internal bedework account) whose preferences are used to set up the skin, default view, view period, views and subscriptions.

Personal & Group Calendaring

Personal and group calendaring is the second calendaring space provided by Bedework. It is the realm of desktop calendar clients, personal schedules, shared calendars, meetings, invitations, to dos, and so on. Much of the calendaring standards focus on this aspect of calendaring, and there are many commercial examples in this space (e.g. Microsoft Exchange with Outlook).

Many of the mechanisms used by the personal calendars, such as preferences, views and subscriptions, are also used by the public user clients to configure their appearance. The major differences between public and personal events lie on the way events are shared and access is managed. A further major difference not fully handled in this release is the need for multi-language support in public events, a need nowhere near as pressing in personal and group calendaring.

Personal calendars also have the problem of interacting with external clients and handling events generated elsewhere, for example through meeting requests.

The personal calendar space is rooted of the main user calendar, named by default "/user". (these names can be configured at build and deployment time to be more appropriate for non-English speaking institutions.)

Below this is a folder for each user of the system for which there has been some activity. Unless an organization chooses to build some form of feed, users who have never had meeting requests and have never logged on will have no presence in bedework. Their user calendars will be created automatically the first time an event is sent to the account or they log on. Personal calendaring is dealt with more fully in Personal Calendars – Page 26

2. DEPLOYING BEDEWORK

2.1 Prerequisites

Requirements

- JDK 1.5
- 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.

Supported databases.

Bedework uses hibernate (http://www.hibernate.org) as a persistence engine. Bedework therefore should run on any database supported by hibernate. In reality, there are some problems, though not insurmountable, with the support of Oracle and MSSql which may require hand-editing of the schema. In future releases we hope to minimize those issues. The current list of hibernate supported databases can be found on their site.

Currently, there are versions of bedework deployed on at least Oracle and MySql5.

Unsupported databases

Because of our use of hibernate, we don't support databases they don't support. In particular MySql4 is known to have problems.

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.

2.2 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 settings mean the system builds and deploys using the files bedework/config/configs/democal.properties and bedework/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" - a caldav server
- "caldavTest" - a caldav test package
- "webapps" - a web client
- "dumprestore" - a dump/restore application
- "freebusy" - the freebusy aggregator

The value of the property "org.bedework.app.<name>.type" corresponds to the name of a subdirectory in bedework/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=webapps
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>
```

defines which class handles administrative groups for the administrative 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 'idealized' 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

2.3 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 or each context 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

Note: if you'd like to set up an initialized database in hsqldb for testing, you can skip to "Build Schema and Initialize" on page 10 and just run "./bwrun schema-export" followed by ".bwrun initdb" against an empty hsql database. To create this, stop hypersonic (if running) and rename (or throw away) the directory named <hsqldb-dir>/demo. Restart hsql and the directory will be recreated in an empty state. See page 10 for more information about the schema-export and initdb commands.

Configure your applications context.xml

Tomcat 5.5.x applications are configured by setting up context.xml files for each application.

Each bedework web application has one or more predefined context.xml files located in the applications war/META-INF directory. Currently all are almost identical and look like this:

```
<Context path="@CONTEXT-ROOT@" reloadable="false">
 <Resource name="jdbc/calDB" auth="Container"
           type="javax.sql.DataSource"
           driverClassName="org.hsqldb.jdbcDriver"
           url="jdbc:hsqldb:hsql://localhost:8887"
           username="sa"
           password=""
           maxActive="8"
           maxIdle="4"
           maxWait="-1"
           defaultAutoCommit="false" />

  <!-- Disables restart persistence of sessions -->
  <Manager pathname=""/>
</Context>
```

The "@CONTEXT-ROOT@" is an example of a token which will be replaced during deployment by a value from the properties file.

To use your own copy of the context.xml file, create such a copy in an accessible location and then update your properties file to refer to it. The relevent property looks like (for example)

```
org.bedework.app.Events.tomcat.context.xml=war/META-INF/publiccontext.xml
```

You probably only need one of these, if you have the above token in your file it will be replaced with the correct context root.

Configure Tomcat 5.5.x

With the above configuration you do not need to configure a database in tomcat's server.xml. However some changes are needed. These are:

- Allow no roles
- Add jdbc drivers
- Allow directory browsing

Allow no roles

Tomcat's handling of security constraints was at some point changed to be absolutely compliant with the servlet specification. Unfortunately, in this regard at least, the specification is somewhat impractical. Within the web.xml security-constraint element is a role-name element which is set to "*" as in this example:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>CalendarAdmin</web-resource-name>
    <description>Public events Administration</description>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>*</role-name>
  </auth-constraint> ...
```

The entry

```
<role-name>*</role-name>
```

used to mean ANY role. It now means by default any role defined in the web.xml. There is no way defined in the servlet specification to have security constraints without roles.

However, it appears that the behavior is configurable in tomcat. In the server.xml there is the definition of the Realm used for authentication. Add a setting for the allRolesMode attribute, in the quickstart it will look like

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
            resourceName="UserDatabase"
            allRolesMode="authOnly" />
```

and you should be able to login without any role assigned.

Add Jdbc drivers

You will need to ensure that the driver jar is added to the common/lib directory.

Allow directory browsing

This change is only required of you are using tomcat to host your stylesheets. Our recommendation is to host them on your public web server. If you do wish to host them on tomcat you must decide whether you want directory browsing enabled or disabled. Enabled allows bedework to discover the stylesheets by checking for the existence of directories. Otherwise marker files must be created. The tomcat default is to disallow directory browsing. If you wish to allow browsing open the file conf/web.xml and find the setting for the listing property:

```
    <init-param>
        <param-name>listings</param-name>
        <param-value>false</param-value>
    </init-param>
```

Change the value “false” to “true” to allow browsing of directories.

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 a defined hibernate SQL dialect such as org.hibernate.dialect.HSQLDialect or org.hibernate.dialect.MySQL5Dialect. The dialect is a class defined on the class path. Hibernate defines a number of 'standard' dialects.

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 dialects included with bedework, look at the org.hibernate.dialect classes in the Hibernate jar file (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 bedework/lib/jdbc and it will appear on the class path for the schema and the dump and restore applications.

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 bedework/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
```

This produces a file named "schema.sql".

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

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

The tables will be created in your database (a file named "schema.sql" is also created). This process is non-destructive, so if you wish to start with a clean database, you will need to

manually drop your tables. 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 initialize an empty database use initdb, e.g.

```
./bwrun initdb
```

This uses an initial data file wrapped up with the dump restore application (data/initbedework.xml). (See also the "restore" target below to restore a different file.)

Note: initbedework.xml creates a super-user "caladmin" with password "bedework". However, there is a potential problem here if you are trying to bring yourself up in your local user address space; you need an administrator that can log into the admin client. To simplify this, we will probably add a property to set the administrative user you wish to use in future releases.

To dump the database data use

```
./bwrun dump <filename>
```

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.

To restore the data use

```
./bwrun restore <filename>
```

where <filename> is the name of a dump file produced by the dump process. The tables must be empty to restore the data.

2.4 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.

2.5 Build and deploy

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.)

```
./ant clean.deploy
```

Do not specify "ant". The form above executes the ant contained in the quickstart which has some extra features installed. If you use the form "ant" you will execute the ant installed in your system which may be an earlier version and/or not have the required libraries.

This will create a number of WAR files in **<bedwork>/dist/** including for example:

```
cal.war, caladmin.war, and ucal.war
```

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.

2.6 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.

2.7 Add calendar suites

Note: You cannot run a public client unless an associated 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 "cal.suite" property in your configuration properties file.

For example, the quickstart defines a calendar "Events" and the property "org.bedework.app.Events.cal.suite" has the value "MainCampus".

1. Within the /caladmin application, create a group which will effectively own the new suite. Only administrative users who are members of this group will be able to administer this suite. For example we can create a group called "myDept" with owner set to your user account and the events owner set to "agrp_mydept"

2. Add users and/or groups to that administrative group.

3. Go back to the main menu, select "Manage Calendar Suites" and add a new suite. Give it a name, set the group, e.g. myDept from above and set the root calendar to be "/public".

4. Give administrators write access. At least add yourself.

5. Log out and log in again and select the group you just created if given a choice.

6. Currently, the system creates a default subscription you need to remove. Go to manage views, select "All" and remove the single subscription. Next, via the main menu, go to manage subscriptions, select that subscription and delete it.

7. Add subscriptions to those public calendars you want to see through one or more views. Set "display" to yes as you add them.

8. Via the main menu, select Manage Views and add any other views you want, adding subscriptions to those views.

9. To set the default view (and other preferences) for the new calendar suite, go to "Manage Preferences" on the main menu. Until you are familiar managing views, you may wish to set the default view to "All"..

2.8 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.

2.9 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.

3. Personal Calendars

Overview.

Personal calendars allow users to carry out all the normal calendaring functions as well as providing a customized view of public events through subscription to public calendars. Currently access to their calendar is through the web clients provided with bedework. In the near future we hope CalDAV will be available giving access to Apple, Sunbird and Outlook users.

Default calendars

A new user will have a set of default calendars created. One of these is the default calendar for events, the name can be configured at deployment but the default is "calendar". In addition a set of special calendars are created, "Trash", "Inbox" and "Outbox". The inbox and outbox are used for scheduling meetings and supporting bedework's implementation of itip. The Trash calendar is used in the usual way to store deleted events before complete removal from the system.

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.

Timezones

Timezones are an important and at times awkward feature of calendaring. An approach a calendar system could adopt would be to ignore timezones completely and display events using local time while storing times as UTC. This loses information implied by the presence of a timezone so bedework attempts to maintain all timezones set and imported by the user.

There is no official registry of timezones. The closest to such a registry is the Olson database which chooses to name timezones according to their continent and nearest large city, for example America/New_York.

System timezones

Bedework provides a set of timezones, the system timezones, which are available to all users of the system. The distributed system comes with timezones derived from the Olson database but any set of timezones could be used. The administrative application provides a means for replacing the system timezones by uploading an xml formatted data file.

Building timezone information.

First we need to convert the Olson database into a set of ics files. The data itself is available from ftp://elsie.nci.nih.gov/pub

The vzic program available via http://www.dachaplin.dsl.pipex.com/vzic/ is used to convert the zone information, Download and unpack the latest source and set the appropriate variables.

We set them as follows:

```
OLSON_DIR=wherever the data was unpacked.
PRODUCT_ID=-//BEDEWORK//NONSGML Bedework Calendar system//US
TZID_PREFIX=
```

The TZID_Prefix can be set to a value which indicates which system the timezone originated from, e.g. "/bedework.org/". Having built vzic (make) and run it (./vzic) a directory named zoneinfo should be built. This is copied into the bwtolls/resources directory.

A copy of this generated data is available at http://bedework.org/downloads/data/

The next step is to convert the data into an xml form for upload. Change directory into the bwtools project and run the following (all one line) which will create a file of xml timezone

information:

```
java -cp bin/bw-tztools-3.2.jar:lib/log4j-1.2.8.jar
org.bedework.tools.timezones.Timezones -dir
projects/bwtools/resources/zoneinfo -f tz.xml
```

A copy of this is generated file is also available at the link above. Finally we need to replace the system timezones in the production system.

Log in to the administrative client as a super user, go to "Upload and replace system timezones", browse to the generated timezones file, then upload.