@enterprise 11.0

Installation and Configuration

April 2024

Groiss Informatics GmbH

Groiss Informatics GmbH

Strutzmannstraße 10/4
9020 Klagenfurt
Austria

Tel: +43 463 504694 - 0
Fax: +43 463 504594 - 10
Email: support@groiss.com

Document Version 11.0.37528

Contents

1 System Requirements

1.1 Platform

@*enterprise* is available for several platforms. For the operation of a server, a Java Runtime Environment (JRE) of Version 17 or higher is required. The following operating systems are supported:

- Windows Variants (2008, 2012, 2016, 2019, 2022, Win7, Win8, Win8.1, Win10, Win11)
- Solaris
- AIX
- Linux

The server should have at least 512 MB of memory for @*enterprise* and 300 MB free disk space.

1.2 Java

To develop @*enterprise* applications, a Java Development Kit (JDK) version 17 or higher must be installed. It is available for download from the Oracle website (http://java.oracle.com) or from another vendor. At the Oracle website, a list of Java ports to other platforms is available.

1.3 Database Management Systems

We support the following DBMSs: Oracle, MS SQL-Server, IBM's DB2, PostgreSQL, Derby, H2. MySQL is supported experimentally.
The following database versions are required:

- Oracle 9i or higher
- MS SQL-Server 2008 or higher

- PostgreSQL V8.4 or higher
- DB2 9.7 or higher on Windows or AIX
- Derby 10.5.3.0 or higher
- H2 2.2.224 or higher
- MySQL 5.0 (experimental)

The database can be installed on the same machine as *@enterprise* or on another networked server.
Somewhat more detailed hints for reasonable configuration of the underlying physical or virtual platforms can be found in appendix Hints for Server Sizing.

1.4 Client

In order to use the the Web-Client, a Web-Browser is all that is needed. Supported products and versions are:

- Chrome 48 or higher
- Firefox 44 or higher
- MS Edge
- Safari 11.0 or higher

2 Installation

2.1 Database Preparation

@enterprise needs a database with one user. In the following we briefly describe the necessary steps for creating a database user for the supported databases.
Please consult the database manuals or the local experts for further information about database setup and creation of a user.

2.1.1 Oracle

You need a database user with the following rights:

```
create session
alter session
create table
create view
```

The user must also have access to a *tablespace* and the permission to add data there.
Example (*EP_USER* is the name of the *@enterprise* database user):

```
create user <EP_USER> identified by <password> default tablespace users;
grant create session, alter session to <EP_USER>;
grant create table, create view to <EP_USER>;
grant unlimited tablespace to <EP_USER>;
```

For full text index creation, the following additional right is needed:

```
grant execute on ctxsys.ctx_ddl to <EP_USER>;
```

If this right is missing there will be errors mentioning 'CTX_DDL' during schema creation or during further schema upgrades. Such errors can be safely ignored. To actually use the full text search capability, the *IndexRefreshTimer* must also be activated.

Since Oracle 11g, a default profile mechanism with resource limitations and password expiration settings might lead to immediate lockout when getting the password wrong or to lockout after a password expiration interval. It is recommended to check the applicable profile parameters and to change them appropriately for the *@enterprise* database user. An unlimited profile can be created with:

```
create profile <EP_UNLIMITED_PROFILE> limit
   composite_limit unlimited
   connect_time unlimited
   cpu_per_call unlimited
   cpu_per_session unlimited
   failed_login_attempts unlimited
   idle_time unlimited
   logical_reads_per_call unlimited
   logical_reads_per_session unlimited
   password_grace_time unlimited
   password_life_time unlimited
   password_lock_time 1
   password_reuse_max unlimited
   password_reuse_time unlimited
   password_verify_function null
   private_sga unlimited
   sessions_per_user unlimited;
```

The specific requirements for your site may vary, in case of doubt check with your local DBA. The profile can be assigned to the user with:

```
alter user <EP_USER> profile <EP_UNLIMITED_PROFILE>;
```

Other useful commands for account administration and trouble shooting are:

- *Check the users profile*:

  ```
  select profile from dba_users
  where username ='<EP_USER>';
  ```

- *Check the properties of the profile*:

  ```
  select resource_name, limit from dba_profiles
  where profile=
    (select profile from dba_users
     where username ='<EP_USER>');
  ```

- *Check the users account state:*

  ```
  select username,profile,account_status,expiry_date from dba_users
  where username='<EP_USER>';
  ```

- *Unlock a users account:*

  ```
  alter user <EP_USER> account unlock;
  ```

- *Unexpire an account or change a users password:*

  ```
  alter user <EP_USER> identified by <password>;
  ```

Hint: If you got the message *Could not get Session ID. Probably no right on V$SESSION*, you have to carry out the following steps in Oracle:

1. *Login as sys*: `sqlplus sys as sysdba`

2. *Assign grant*: `grant select on v_$session to <EP_USER>;`

@*enterprise* can display query plans for queries executed via the reporting component or via the query tool. To enable the functionality, parameters in configuration section "Other Parameters" need to be set, and the database user needs the appropriate privileges to access the information. This can be achieved via:

1. *Login as sys*: `sqlplus sys as sysdba`
2. *Assign grants*:

```
grant select on v_$session to <EP_USER>;
grant select on v_$sql_plan_statistics_all to <EP_USER>;
grant select on v_$sql to <EP_USER>;
grant select on v_$sql_plan to <EP_USER>;
```

Hint: After such a change of privileges, a restart of @*enterprise* is required. Please note that the query plans are also written into the log file and that this functionality is not intended to be permanently switched on in production systems. It can be enabled by activating the parameter *database.queryplan.get* in section *Other Parameters* in the configuration. The queries in property *database.queryplan.query.oracle* might need some adaptations depending on your version of Oracle.
If the use of full-text search or WfXML2 functionality is intended with Oracle as the underlying DBMS, you must select the Oracle LOBs database type in the configuration (and not the legacy mode with Oracle LONGs). Since Oracle supports just one LONG column per table, the tables for WfXML2 functionality will not be generated when LONGs are used instead of LOBs.

Oracle offers the possibility to set the semantic of varchar/varchar2 datatypes (BYTE or CHAR). The decision of setting the correct type could be necessary, if UTF-8 texts should be stored. An example could be that a field has a length-restriction of 100 characters and the text to be stored contains 100 characters with 2 umlauts. Because of UTF-8 encoding the text will grow up to 102 Byte and could not be stored anymore.
For this purpose you can change the semantic on two ways:

- global by using following statement:

```
alter system set nls_length_semantics='CHAR' scope=both;
```

- per session (db-session-environments in @*enterprise* configuration) by using following statement:

```
alter session set nls_length_semantics='CHAR';
```

Hints for the performance of Oracle-based @*enterprise* installations can be found in appendix Database Performance Hints under Oracle.

2.1.2 MS SQL-Server

@*enterprise* requires a case insensitive installation of MS SQL-Server.
We generally recommend one SQL-Server database per @*enterprise* installation, but for testing or development purposes, several installations can use the same database, provided that proper user schema separation is implemented (i.e. each installation gets its own user and a dedicated schema).
When creating a SQL-Server database, use the option 'ANSI NULL is default'. You can specify it in the database property panel or by execution of a stored procedure after installation.

```
ALTER DATABASE <dbname> SET ANSI_NULL_DEFAULT ON;
```

<DBNAME> must be replaced with the name of your database. The procedure results in behavior consistent with the ANSI standard regarding the handling of NULL values.
The database user for @*enterprise* must have the right to create tables, for example via the role db_owner.

It is advisable to use Statement-Level Snapshot Isolation in order to avoid shared locks by readers. Enable it with:

```
ALTER DATABASE <dbname> SET READ_COMMITTED_SNAPSHOT ON;
```

Note that no other users are permitted to access the database when you issue this command and that the feature is available only in SQLServer 2005 or higher.
If you use full-text search, please ensure that MSSEARCH service is running and automatic population (for creating indices of the full-text catalog) is activated.
In the following we present a script that creates a database and all the needed security and schema context:

```
USE master
GO
CREATE DATABASE epdb
GO
ALTER DATABASE epdb SET ANSI_NULL_DEFAULT ON
GO
ALTER DATABASE epdb SET READ_COMMITTED_SNAPSHOT ON
GO
CREATE LOGIN eplogin WITH
       PASSWORD='eppasswd',
       DEFAULT_DATABASE=epdb,
       CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO
USE epdb
GO
CREATE SCHEMA epschema
GO
CREATE USER epuser FOR LOGIN eplogin WITH
       DEFAULT_SCHEMA=epschema
GO
ALTER ROLE DB_OWNER ADD MEMBER epuser
```

```
GO
ALTER AUTHORIZATION ON SCHEMA::epschema to epuser
GO
```

Just change the identifiers starting with `'ep'` to values that make sense in your context (and of course provide a reasonable password as well).
Please note that SQL-Server has two related, but not identical notions of "login" and of "user". It is the login name (eplogin) you created with "CREATE LOGIN" that should be entered in the "Database Userid" field when connecting to the database.
While this is a usable starting point, there are numerous unspecified options which take their default values. In a production environment there might be special requirements for e.g. for the physical storage parameters, for permissions of the user or the kind of login (Windows or SQLServer) you will be using.

2.1.3 DB2

When using DB2 you have to create an operating system user. Afterward a database user is created with the rights *connect to database* and *create tables*. Set the character set of the database to UTF-8 and the standard size of the buffer pool and table space to 16 KB. Then you create a database schema, for which the user is authorized.

2.1.4 PostgreSQL

Installation of PostgreSQL can vary a bit depending on the underlying platform.
For Windows, an installer is being used. During installation of PostgreSQL choose at least the following components:

- Database Server
  - Data Directory
  - National Language Support
- User interfaces
  - psql
  - pgAdmin III (optional admin GUI)
- Database Drivers : JDBC Driver

The windows-installer will display a "initialize database cluster" dialog, it is advisable to use `UTF-8` as encoding and to check "accept connection on all interfaces" if remote connections to the database are needed.

In case of Linux as underlying platform, we recommend to use the package manager of your distribution to install the `postgres` package. Database creation and listener/interface specification will usually have to be performed manually afterward; the steps will be described below as optionally.

Independent of the used platform (Windows/Linux), in the `data` subdirectory of the installation directory, edit the `pg_hba.conf` file to allow access from remote machines if desired, e.g.:

```
host    all     all     10.10.10.0/24      md5
```

In the `data` subdirectory of the installation directory edit the `postgresql.conf` file. Make sure that parameter `default_with_oids` is turned off:

```
default_with_oids = off
```

It is advisable to restrict the search path to the current user schema with:

```
search_path = '"$user"'
```

Extensive PostgreSQL logging can pose problems regarding memory consumption. Please make sure that `log_statement` is set to either `off` or `ddl` and that `log_min_duration_statement` is deactivated (use a value of `-1`).
Optionally make sure that postgres listens on the appropriate network interfaces, if remote connections are needed:

```
listen_addresses = '*'
```

Then restart the postgres service:

- in Windows: via the service manager
- in Linux: via the appropriate command for the init framework of your distribution (e.g.: `systemctl restart postgresql`)

Then, as `postgres` user, the PgAdmin III gui or the `psql` command line tool can be used to

- Create a User Account ("Login Role" in Postgres Terminology)

  ```
  create role <EP_USER> login password '<eppasswd>'
  noinherit valid until 'infinity';
  ```

- Optionally create a database if no one was created at Postgres installation time, or if you want a separate one for *@enterprise*:

  ```
  create database <DBNAME> encoding='UTF8';
  ```

- Connect to the created database, either via the PgAdminIII gui or with psql:

  ```
  \c <dbname>
  ```

- Create a Schema: (preferably without any schema name, so the name of the schema will be the same as the name of the login role):

  ```
  create schema [<SCHEMANAME>] authorization <EP_USER>;
  ```

- Provide an appropriate default schema: If the names of the schema and of the login role must differ in your installation, be sure to

- either set the default search path for the login role in the database via:

```
alter role <EP_USER>
in database <DBNAME>
set search_path = <SCHEMANAME>;
```

- or set the search path for sessions in the Session Environment field in install step 10 in section Extract and Install:

```
set search_path=<SCHEMANAME>
```

To activate support for the soundex search, use the following command (the `fuzzystrmatch*.sql` files may be located in a separate package named `postgres-contrib`).

```
create extension fuzzystrmatch schema <SCHEMANAME>;
```

2.1.5 Derby and H2

Derby and H2 are embeddable, small footprint database engines written in Java.

H2 is perfectly suited for development purposes and test deployments. Derby is also well suited for development and test activities. But even in such constellations, concurrency issues (in particular: deadlocks) may arise with Derby.

For heavyweight multiuser installations the use of either Derby or H2 is not really advisable.

Neither Derby nor H2 do need any special preparation.

H2 consists of a single *h2-<version>.jar* file. For Derby, the three files *derby.jar*, *derbyshared.jar* and *derbytools.jar* are needed.

2.2 Extract and Install

This section describes how to install the *@enterprise* stand-alone server. *@enterprise* can be downloaded from our website and is packed in one single file named `ep-setup-11.0.xxx.jar`. The installation can be started with a double-click on the downloaded file. If the jar file cannot be executed with a double-click you need to call *java -jar ep-setup-11.0.xxx.jar* in a terminal to start the installation. In any case Java JRE 17 (or higher) is required on your system.
The setup process consists of the following steps:

1. Verify if this is the version of *@enterprise* that you want to install and start the setup by clicking on *Ok*.
2. Installation directory: The directory where the system will be installed.
3. Specify the directory of the Java compiler and interpreter.
4. Choose the port on which the *@enterprise* server will run.

5. If your server operating system is MS Windows you can install a service; details can be found in section Installing as a Windows Service.

6. Now setup shows you information about how you can start the server and continue the setup process. If the checkbox *Start epstart.bat and the browser now* is activated the setup will try to start the server and open a browser for you. If this fails and if you did not install a service, you have to start the server manually by executing the batch or shell file (epstart.sh or epstart.bat). If your browser didn't already do it, please navigate to `http://localhost:port/`, where *port* is the port number that you have chosen during the previous setup steps. The rest of the installation is done with the browser.

7. The first screen is the Welcome-screen, click on *Start Setup* to start the configuration.

8. On the next screen you specify a logical name for the server (server ID), the license key and the server's default language. Please note that changing the server id will result in invalid sessions after the first server restart (this applies to installations using the internal Jetty web server; not to installations within an application server).

9. Now you can load a database JDBC driver. Use the *JDBC Driver Help Page* for information about different databases and their JDBC drivers.

10. On the next screen you have to specify some database parameters. We suggest to use the help function (the question mark next to *Database Type*) to fill the Database Type, JDBC Driver Class, and JDBC URL fields with valid values.

    - Database Type: The database; you can select ORACLE, DB2, MS SQL-Server, Postgres, Derby or H2
    - JDBC Driver Class: Java class that contains the driver. Take a look at the table on page 40 for a list of driver classes.
    - JDBC URL: URL for the database. The syntax of this string depends on the JDBC driver used. See the examples on page 40 or consult the documentation of the driver. *@enterprise* allows to configure data sources too. For further information take a look in chapter *@enterprise* and Datasources.
    - Credentials not needed: Activate this checkbox, if database connection without credentials (user-id/password) should be used when authentication is accomplished externally (e.g. SQLServer Windows native authentication).
    - Database Userid: The ID of the user with whom you want to connect to the database.
    - Database Password: Password for the database user with the ID that you entered above.
    - Number of Connections: Default number of database connections.
    - Session Environment: You can specify SQL-commands, which are executed for each connection after connecting, for example: `set TEXTSIZE 10000000` (SQLServer) or `set search_path=ep` (PostgreSQL)

    If SQLServer Windows native authentication is used, the following steps are needed:

- If using sqljdbc driver, the file *sqljdbc_auth.dll* must be available in *@enterprise* root directory (or in case of service in <ep_root>/services directory).
- Add to driver url the string `integratedSecurity=true`, e.g. jdbc:sqlserver://<host>:<port>;DatabaseName=<dbname>;integratedSecurity=true
- Activate checkbox *Credentials not needed* and perform next setup step.

11. Now the database and driver will be tested. Optionally you can test if your database can store Unicode characters.
12. The next step is the creation of the database tables. The time may vary depending on your server's speed and the database that you use. If a schema of a (previous) *@enterprise* version exists, setup cannot be continued at this point!
13. After initializing the database, some internal services have to be started.
14. On the next screen the password of the system administrator can be specified.
15. Now a user and an organizational unit can be created. The following roles will be given to this user: *all*, *home* in the inserted organizational unit, and *sys*.
16. If desired, it is possible to load examples such a demo application and standard reports.
17. Congratulation! You finished the setup of *@enterprise*. Click on *Login* to go to the login page, where you can immediately start to use *@enterprise*.

By completing the previous steps you finished the setup of *@enterprise*. If you want to change the configuration or configure advanced settings, take a look at chapter Configuration.

Installing *@enterprise* in command line mode

Setup can also be started from the command line:

```
java -jar ep-setup-11.0.xxx.jar . default
```

The mandatory arguments are the destination directory and Java location (*default* can be used for the default Java version). Optional arguments are the http port (default is 8000) and the windows service name (no default). However, if your server operating system is MS Windows and you want to install a service, you have to enter either *0* for the http port or to define a concrete http port, e.g.:

```
java -jar ep-setup-11.0.xxx.jar . default 0 <myservicename>
```

After extracting the files please navigate to `http://localhost:port/`, where port is the port number that you have chosen during the previous step (or 8000 if the port was optional or *0*). The rest of the installation is done with the browser and is described from step 7.

2.2.1 File structure of stand-alone server

In the installation directory of @*enterprise* you will find the following three subdirectories:

- **base**: this directory holds the @*enterprise* software parts, e.g. all the libraries needed to run the @*enterprise* server or the batch/shell script file used to start the @*enterprise* server. The content of this directory will only be changed when upgrading the @*enterprise* software.
- **local**: this directory serves as the container for all application/project specific software parts and is organized in the following directories and files:
  - **appls**: this is the default root directory for all installed application.
  - **conf**: when splitting the @*enterprise* server configuration into multiple configuration files the parts that will normally not be changed. when running the @*enterprise* server should be placed here.
  - **lib**: additional application independent libraries needed to run the @*enterprise* server shall be located here, e.g. the database driver file uploaded in the installation process is stored in this directory.
  - **localjavaargs**: is meant to hold Java properties applied to the Java call starting the @*enterprise* server, e.g. setting the minimum heap size.

  These directories and files should only change when installing new or upgrading existing applications, as well as when project specific needs change.
- **var**: this section will hold all the data that is specific for a single @*enterprise* server instance and is created or changed when running that instance. It consists of the following directories and files:
  - **appls**: for each installed application this directory will contain a subdirectory with the application's Id as its name containing the configuration file `appl.prop` of that application.
  - **conf**: holds the default configuration file `ep.conf` of the @*enterprise* server or, when splitting the @*enterprise* server configuration into multiple configuration files, should be the place where those configuration files are located which properties are meant to be changed during the operation of this server instance.
  - **forms**: default directory for the form's class files generated by the @*enterprise* server instance.
  - **log**: default directory for the access and error log files of the @*enterprise* server instance.
  - **tmp**: default directory for temporary files created when running the @*enterprise* server instance
  - **varjavaargs**: is meant to hold Java properties specific for this @*enterprise* server instance which will be applied to the Java call starting the @*enterprise* server, e.g. activating some debugging stuff.
  - **WEB-INF/web.xml**: in a standalone installation, the web.xml file is first searched for here, and if not found the one from `base/WEB-INF/web.xml` is being used.

This directory structure reflects the responsibilities within an @*enterprise* installation very clear:

- **base**: the @enterprise software
- **local**: the project software
- **var**: the runtime data

Customization

By default the directories `base`, `local` and `var` are all located in the @*enterprise* installation directory. This can be customized using environment variables. Also the configuration files can be customized in such a manner. Here is an example in which we define a script file `startup.sh` in which those customizations a made:

```
#!/bin/sh
EP_BASE="/ep/server/ep11"
EP_LOCAL="/ep/local/ep11"
EP_VAR="/ep/var/ep11"
EP_CONFFILES="$EP_VAR/conf/ep.conf,$EP_LOCAL/conf/fixed_ep.conf"

export EP_BASE
export EP_LOCAL
export EP_VAR
export EP_CONFFILES

sh "$EP_BASE\epstart.sh" %1
```

In situations where you cannot or do not want to use environment variables you can customize those directories using the following Java properties:

- `ep.base`
- `ep.local`
- `ep.var`
- `ep.conffiles`

But be aware: our script files do set default values for the environment variables if not set already and the @*enterprise* server prioritizes the values of the environment variables before the values of the properties. So this is only feasible if you perform the java call starting the server on your own.
In the example above we additionally split up the @*enterprise* server configuration into the following files:

- `$EP_VAR/conf/ep.conf`: is meant to hold the configuration properties being changed when running the server
- `$EP_LOCAL/conf/fixed_ep.conf`: holds configuration properties which should be part of the delivery and should not be changed later on

2.2.2 Bootstrap in stand-alone server (Jetty)

We use a bootstrap mechanism which automatically builds the classpath. This mechanism allows to keep the batch- and/or shell-file simple and clear.
Additional elements for the classpath can be conveniently specified via the environment variable `EP_BOOTSTRAP_PATH` or via the java property `-Dep.bootstrap.path` with the following special behavior:

- `classes`: all files within this folder are added to classpath
- `lib`: all files within `lib` with the extension `jar` are added to classpath
- `mylocal.jar`: the corresponding file is added to classpath
- all other paths are scanned for a `classes` and `lib` directory and the corresponding entries will be added to classpath. If these directories are not available, the entered directory will be added to the classpath.

Hint: The first entered path (leftmost) of property `-Dep.bootstrap.path` is loaded first, the rightmost path is loaded at last. The jar-files of the `lib` directory are loaded in alphabetical order.

Example for an entry in `var/varjavaargs`:

```
-Dep.bootstrap.path=C:/eproot;C:/extension/classes;../libs/lib;C:/myjar.jar;.
```

- `C:/eproot` is scanned for a `classes` and *lib* directory
- `C:/extension/classes` is added to the classpath
- `../libs/lib` results in adding all contained jar files to the classpath (scanned relative to the base-path)
- `C:/myjar.jar` is added to the classpath
- `.` means, that the base-path is scanned for a `classes` and `lib` directory. If these directories are not available, all elements of the base-path will be added to the classpath.

2.3 Installing as a Windows Service

In Windows you can configure a stand-alone installation of @*enterprise* to run as a service. This can be done while installing (see the previous section) or later by calling the `epservice create` script in the `base/service` subdirectory of @*enterprise*
@*enterprise* uses the *svcbatch* framework from `https://github.com/mturk/svcbatch`.

Hint: When using this mechanism, do not use any whitespace characters in path specifications or directory names.

2.3.1 Components of the Framework

The service framework consists of the following files in the `base\service` subdirectory:

Common service framework files

- `svcbatch.exe`: The core process runner.
- `Elevate32.exe, Elevate64.exe`: Admin rights utility for Installations with User Account Control (UAC); c.f. `sudo` in Linux.
- `epservice.bat`: Used to create, control and delete the service. Can be called in any of the following modes:
  - `epservice create`: Creates the service. This is the only variant that is being called (during initial installation). All other variants are for manual usage via the command line.
  - `epservice delete`: Deletes the service.
  - `epservice dump`: A thread dump of a service is written to its log files.
  - `epservice rotate`: Rotates the log files of the service.
  - `epservice start`: Starts the service.
  - `epservice stop`: Stops the service.
  - `epservice status`: Displays the service status.

  All those calls must be performed with the proper permissions when UAC is enabled, i.e. prefix the calls with `ElevateXX`. Deleting or changing a service should only be attempted when the service is not running currently. It is also recommended to abstain from using/opening either the service manager or registry editor when creating or changing the service.

Other files to be considered are:

- `environment.bat`: This file is created during the extraction of *@enterprise* from the ep-setup-11.0.xxx.jarfile and placed in the `var\service` subdirectory. It contains the id of the service. The file has the following form:

  ```
  set "SERVICE_NAME=@enterprise110"
  ```

  This file can also be created manually when you abstained from installing *@enterprise* as a service during initial installation and want to deploy it as a service later on. In fact, it will be automatically created if it does not exist and `epservice` is called like `epservice <operation> <servicename>`
- `aftercreation.bat`: Additional actions for the service definition (to be carried out) after the creation like dependencies on other services or account details will have to be carried out manually (via the services console in Windows), or by creating a `aftercreation.bat` file in the `var\service` subdirectory. This (optional) script will be called during `epservice create` and could look like this:

```
sc "%SERVICE_NAME%" config obj= accountname password= password
sc "%SERVICE_NAME%" config depend= aservice/bservice
```

- `var\service\log`: location of the log files of *svcbatch*
- `*javaargs`: Additional arguments for the Java virtual machine should be specified in the `local\localjavaargs` or `var\varjavaargs` argument files.[1].

If you want to customize the file structure of the service installation like described in Customization, you have to

- set the appropriate environment variables `EP_BASE`, `EP_VAR`, `EP_LOCAL` and `EP_CONFFILES`,
- assure that the service name is defined in `%EP_VAR%\service\environment.bat`

```
set "SERVICE_NAME=@enterprise110"
```

- delete the service via `epservice delete`
- recreate it via `epservice create`

The values of the environment variables are preserved in the service definition in the registry entry `HKLM\SYSTEM\CurrentControlSet\Services\<service_name>\Environment`; they can also be changed via `regedit`.

2.4 Installing as a Linux Daemon

For `systemd` based Linux distributions, there is a quite simple pattern to use *@enterprise* as a daemon without any additional overhead. We provide a template unit file `enterprise.service` located in the `service` subdirectory of the *@enterprise* installation. Copy this file to your local `systemd` directory (usually at `/usr/lib/systemd/system`). Modify it according to your needs:

- replace `%epuser%` with the user id of the Linux user[2] that should run *@enterprise*.
- replace `%epworkdir%` with the absolute canonical path[3] to the *@enterprise* installation directory.
- replace `%epjava%` with the absolute canonical path to the Java executable (be sure to keep the `/usr/bin/env` preceding it).
- state appropriate dependencies on other services for startup (e.g. dependency on DBMS) in the `Requires` line.
- adapt the arguments of the command line (be sure to use ␣\) at the end of each continued line). Additional arguments for Java should be specified within the `local/localjavaargs` or `var/varjavaargs` argument files.

[1] The values in those files are read when starting the service or when restarting @enterprise
[2] please note that privileged ports (port numbers below 1024) can only be opened when run with `User=root`.
[3] such a path starts at / and does not follow any symbolic links. Its pathname should not contain any spaces.

Your `enterprise.service` unit description file could look like this:

```
[Unit]
Description=@enterprise BPMS
After=syslog.target network.target
#Requires=a.service b.service

[Service]
User=epadm
Group=users
Type=simple
Restart=on-failure
RestartForceExitStatus=2
Environment='EP_BASE=./base'
Environment='EP_LOCAL=./local'
Environment='EP_VAR=./var'
Environment='EP_CONFFILES=./var/conf/ep.conf'
WorkingDirectory=/opt/ep/ep11
ExecStart=/usr/bin/env /usr/lib64/jvm/java-17-openjdk/bin/java \
@${EP_BASE}/epjavaargs \
@${EP_LOCAL}/localjavaargs \
@${EP_VAR}/varjavaargs \
-jar ${EP_BASE}/lib/ep-bootstrap.jar

[Install]
WantedBy=multi-user.target
```

After changing the file, its recommended to reload the systemd daemon by means of

```
systemctl daemon-reload
```

The service can then be administrated with the usual systemd stanzas:

- enable startup of @*enterprise* at system startup:

  ```
  systemctl enable enterprise
  ```

- disable startup of @*enterprise* at system startup:

  ```
  systemctl start enterprise
  ```

- enquire the state of @*enterprise*:

  ```
  systemctl status enterprise
  ```

- start @*enterprise* manually:

  ```
  systemctl start enterprise
  ```

- stop @*enterprise* manually:

  ```
  systemctl stop enterprise
  ```

- restart @*enterprise* manually:

  ```
  systemctl restart enterprise
  ```

2.5 Using an Application Server or Servlet Container

If you want to run *@enterprise* in an application server (e.g., Oracle's *WebLogic*) or a servlet container (e.g., Apache's *Tomcat*) you need the *@enterprise* web application archive file named `ep110.war` which is especially prepared for this purpose. Deploy this file in your server. Afterward, open your browser and navigate to `http://host:port/context-root/`, where `host` and `port` must be the right values for accessing your server and `context-root` is the context root that you chose when deploying the file. See section Extract and Install, starting with step 7 for details about the rest of the installation.

Hint: When Using Derby or H2, the database name of the JDBC-URL may be specified relatively!
In Derby in embedded mode, the 'ep' part in *jdbc:derby:ep;create=true*, is relative to the current directory or relative to the directory specified in the *derby.system.home* system-property (if this is present).
In H2 in embedded mode, the './ep/epdb' part in *jdbc:h2:./ep/epdb;DB_CLOSE_ON_EXIT=FALSE* is either relative to the current directory or relative to the directory specified in the *h2.baseDir* system-property (if this is present).
In a scenario of deploying multiple *@enterprise* systems as different web-applications in one servlet-container, with each of the systems using a dedicated embedded Derby or H2 instance, use unique path names to the database files per web-application
(e.g. for Derby *jdbc:derby:databases/app1/derbydb;create=true* and *jdbc:derby:databases/app2/derbydb;create=true*).

Hint: Uploading of driver jar files during installation might not work satisfactorily depending on classloading implementation details and preinstalled driver jar files in some common area of the application server. If you want a specific version of a driver, it is advisable to change the underlying driver jar in the application server. If this is not feasible, try to include the desired driver jar file in the `ep110.war` file.
This might not be sufficient for your particular application server. E.g. since Weblogic comes with its own version of Derby, for the combination of Weblogic as application server and Derby as database management system it is also required to specify the class loading order by adding the `org.apache.derby.*` packages name to the `prefer-application-packages` element located in `WEB-INF/weblogic.xml` file in the `ep110.war`.

Hint: If Tomcat is used as Servlet Container and UTF-8 encoding should be used, you have to set following attributes in Tomcat's *server.xml*:

- URIEncoding="UTF-8"
- useBodyEncodingForURI="true"

Typically:

```
<Connector port="8080"
 maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
 enableLookups="false" redirectPort="8443" acceptCount="100"
 debug="0" connectionTimeout="20000"
```

```
URIEncoding="UTF-8" useBodyEncodingForURI="true"
disableUploadTimeout="true" />
```

Hint: The servlet API version 4.0 is being used in combination with the traditional namespace (`javax.*`) of JavaEE 8. Compatible appserver versions must be used (Tomcat 9.X, Jetty 10.x, Weblogic 14.X).

2.5.1 File structure in an Application Server or Servlet Container

In this case there is a tiny difference in the file structure compared to the stand-alone installation: the directory containing the *@enterprise* software parts is named `WEB-INF` instead of `base`.
Also customization is a bit different to the stand-alone case:

1. Files `localjavaargs` and `varjavaargs` will not be considered
2. Only the location of directories `local` and `var` can be customized, the base directory is always `WEB-INF`
3. Names of the environment variables or java properties must have a proper suffix to distinguish between several different deployments within the same container. We use the context root of the web app to discriminate between the locations. In the case of environment variables the suffix of the variable name is `_<CONTEXT>` in uppercase. For java properties it is `.<context>` in lowercase. e.g. `EP_VAR_EP110` respectively `ep.var.ep110`. So if you have two deploments in the same container, the first one in context root `ep110dev` and the second one in `ep110test`, a customization might look like this.

   ```
   export EP_VAR_EP110DEV=/opt/wf/ep110dev/var
   export EP_LOCAL_EP110DEV=/opt/wf/local
   export EP_CONFFILES_EP110DEV=${EP_VAR_EP110DEV}/conf/dev.conf,\
   ${EP_LOCAL_EP110DEV}/conf/fixed_common.conf

   export EP_VAR_EP110TEST=/opt/wf/ep110test/var
   export EP_LOCAL_EP110TEST=/opt/wf/local
   export EP_CONFFILES_EP110TEST=${EP_VAR_EP110TEST}/conf/test.conf,\
   ${EP_LOCAL_EP110TEST}/conf/fixed_common.conf
   ```

The details of the customization via environment variables or java properties are naturally dependent on the Application Server or Servlet Container. For example in Apache Tomcat 9.x the environment variables can be added to a file named `setenv.bat` respectively `setenv.sh` which needs to be added to directory `bin` of the Tomcat installation.

2.6 Unattended installation and upgrade

@enterprise offers the possibility to create or upgrade a complete installation without additional user interaction. This could be necessary, if many *@enterprise* systems must be deployed with the same installation files, but with different configuration settings.

Example: A test system, a staging system and a production system are needed. All systems have the same applications to deploy, but with different database and mail-server settings.

As point of origin the setup files (**ep-setup-11.0.xxx.jar** for standalone or **ep110.war** for application server) provided by Groiss Informatics GmbH should be taken and enriched with own files for configuring/installing *@enterprise* itself and its applications. The following sections describe the preparation and deployment of such an installation.

2.6.1 Preparation of installation files

As point of origin the following artifacts are needed:

1. Depending on the usage of *@enterprise* as standalone server (Jetty) or in an application server (see section Using an Application Server or Servlet Container) the appropriate setup file is needed.
2. The database driver file.
3. The local configuration of *@enterprise* (see section Definition of the configuration for details).
4. The installable, unpacked applications (not within a ZIP file) whereby the folder name must consist of the application id.
5. Other needed files for installation, e.g. application-independent XML-imports or csv-files (application-dependent files should be part of application zip files).
6. A groovy-script file which performs the installation of the applications (see section Definition of an install script for details).

The files of items 2-6 must be added to the appropriate setup file with a ZIP tool or the jar-command of the JAVA distribution as shown in following examples:

```
jar -uf ep-setup-11.0.xxx.jar local var
jar -uf ep110.war local var
```

File structure for standalone installation (Jetty):

```
local
 appls
   applid1
     classes
     lib
     appl.prop
   applid2
     classes
     lib
     appl.prop
   ...
   install.scr
 lib
   db_driver.lib
```

```
var
 conf
   ep.conf

ep-setup-11.0.xxx.jar
```

File structure for installations in application server:

```
 local
  appls
    applid1
      classes
      lib
      appl.prop
    applid2
      classes
      lib
      appl.prop
    ...
    install.scr
  lib
    db_driver.lib
 var
  conf
    ep.conf

ep110.war
```

2.6.2 Definition of the configuration

For the configuration a file with name `ep.conf` has to be created in the `var/conf` directory. It should contain the parameters for *@enterprise* while the application specific parameters need to be defined in the appropriate `appl.prop` files of the applications (within `local` directory).

Mandatory parameters for *@enterprise* are

- `ep.autodeployment.enabled`: This parameter with value *true* is needed to indicate, that an unattended installation should be performed.
- `setup`: In case of an unattended installation this parameter must be set to value *true* to avoid starting the SETUP wizard in Browser.
- `avw.license`: A valid license key (see section License).
- `avw.servername`: The name of the server which is stored in the *@enterprise* server object.
- Database parameters: At least the set of parameters as defined in the example below (see section Database for details).

- `httpd.port` for standalone installations: The port on which the standalone server runs (see section HTTP server for details). This parameter is not needed for installations in application server.

Mentionable optional parameters for *@enterprise* are

- `ep.autodeployment.applications.sequence`: If the installation/upgrade order of the applications is important, an accordingly sorted comma separated list of application-ids can be specified with this parameter. If this parameter is not given, all applications of directory *appls* will be installed in alphabetical order of the folder names.
- `ep.server.hostname`: The optional host name of the server which is stored in *@enterprise* server object. If this configuration parameter is not given, the host name is determined automatically and stored in the *@enterprise* server object.

Example for a configuration file (ep.conf):

```
# required parameters
ep.autodeployment.enabled=true
setup=true

avw.license=<license_key>
avw.servername=ep11

database=com.dec.gi.sql.DBOracleLOB
database.driver.class=oracle.jdbc.OracleDriver
database.url=jdbc\:oracle\:thin\:@localhost\:1521\:mydb
database.user=<ep_user>
database.password=<ep_pwd>
database.connections.max=8

httpd.port=8090


# optional parameters
ep.autodeployment.applications.sequence=applid2,applid1

logger.trace=DEBUG
httpd.maxthreads=25
httpd.minthreads=2
Locale.list=en_US,en_GB,de_AT
Locale.language=en
Locale.country=GB
ep.product.name=WFM system
passwdpolicy.days_password_valid=180
passwdpolicy.days_warning_before=10
passwdpolicy.max_count_invalid_logins=5
passwdpolicy.min_length=8
passwdpolicy.min_capitals=1
passwdpolicy.min_digits=2
passwdpolicy.min_others=1
passwdpolicy.history_steps=10
```

2.6.3 Definition of an install script

All other operations in *@enterprise* which cannot be handled within the application configuration (e.g. import csv-files, import test data, etc.) can be achieved with a groovy-script file within the `appls` directory. The file must be named `install.scr`! The groovy context is the same as for the administration shell component (see Administration Guide - section *12 Administration Shell*).

Example for a script file:

```
//create test org-unit
testOU = orgdata.createOrgUnit();
testOU.setId("testOU");
testOU.setName("Test-OU");
testOU.setActive(true);
orgdata.insert(testOU);

//install test user for application "appl1"
admin.importXML("applid1/test/testusers.xml");
```

2.6.4 Performing the installation

The installation of a complete packaged setup file can be achieved in following ways:

- Standalone: Perform the JAVA call in command line

  ```
  java -Djava.awt.headless=true -jar ep-setup-11.0.xxx.jar <dest_dir> <java_dir>
  ```

  The <dest_dir> is the destination directory where *@enterprise* should be extracted. The <java_dir> defines the location of Java installation directory; if the keyword "default" is entered, the default location of JAVA is taken.

- Application server: Deploy the WAR-file in application server.

For both ways in the first step the content of the setup file is extracted.
In a further step the automatic deployment service is started and performs the installation or - if a deployment already exists - the upgrade operation. The procedure is always the same: First *@enterprise* itself is installed/upgraded and subsequently the applications. Finally the script file `install.scr` is executed.
After successful installation/upgrade operation the server log contains the message "Initialize/upgrade operation with automatic deployment is successful."
In case of standalone installation (Jetty) the server is restarted automatically (if needed). In case of an application server it must be restarted manually (also written as info in the server log)!

2.6.5 Upgrade considerations

The recommended way to perform an upgrade is a re-deployment by performing the following steps:

1. Create a new installation directory

2. Call the command defined in Performing the installation with the new directory as the destination directory
3. Stop the server
4. Copy all files and directories which you want to keep from the old `var` directory into the new one. Normally this is only needed for configuration files (and the database directory if H2 or Derby are being used in embedded mode).
5. Start the server

In case of running *@enterprise* within an Application Server a re-deployment is the only way to perform an upgrade, but the steps are a bit different than sketched above:

1. Stop the Application Server
2. Backup your old installation directory
3. Replace the WAR file in the Application Server
4. Start the Application Server
5. When the deployment is finished stop the Application Server
6. Restore all files and directories which you want to keep from the old `var` directory into the new one. Normally this is only needed for configuration files (and the database directory if H2 or Derby are being used in embedded mode).
7. Start the Application Server
8. Only needed if you did copy the database directory into the new `var` directory: restart the Application Server as soon as the web application is available (i.e. the database upgrade has been performed).

2.7 Basic considerations for backup and recovery

According the operational aspects of backup and recovery, an *@enterprise* installation comprises of the following component types:

- Basic *@enterprise* software artifacts
- Application specific software artifacts
- Configuration data
- Application Logfiles
- Database content

For the first two types which are located in directories `base` respectively `local`, quite ordinary backup and recovery measures are perfectly appropriate. The small volume and infrequent changes to these components allow for periodic full backups of these directories.
Configuration data comprises a small set of very small configuration files (`ep.conf` in the `var/conf` directory and `appl.prop` in the application directories most likely located in `var/appls`). Changes to those files will be relatively infrequent (after an initial production phase), but might be critical to proper operation. Frequent or even immediate backup of those files is recommended, possibly by incorporating them in a version control system.
Application log files (usually in the `log` subdirectory of `var` area) should be rotated and a periodic or rotation triggered copy could be made. The log files are not essential for the operation, so there is no need to recover them. Nevertheless, they might be essential for gaining insight of the nature of the problem and help to avoid further errors.
Concerning the database, backup and recovery measures are obviously vendor specific. But some general remarks are nevertheless applicable. Periodical backup of the data files is strongly recommended. To be able to recover to the youngest point in time possible, transaction log writing must be enabled. The logs must be switched and shipped to a safe location on the fly.
Concrete recovery measures depend on the type and range of the disaster, but in general, they consist of recovering the database from the latest backup of data files and transaction logs, to copy the software artifacts of *@enterprise* and of the application back to their destinations and to reinstitute the configuration data.

3 Configuration

3.1 General Aspects

3.1.1 Basics:

This chapter describes advanced configuration parameters of *@enterprise*. You can change the data that you entered at setup as well as additional configuration here. Open the configuration area in the system administration by clicking on *Configuration* in the menu on the left side.

In order to save your changes, you must use the *Save* icon in toolbar, which is available on every configuration page. After activating this button, the changes are stored in the file *ep.conf* by default, which can be found in the folder *var/conf* of *@enterprise* installation directory. The location of the configuration file(s) can be changed via the environment variable `EP_CONFFILES` or the java system property `ep.conffiles`.
When changing settings via GUI, no server restart is necessary, except when the notification icon appears (yellow triangle)!

Each parameter has a value which is set by default. If the entered value is different to the default value, an icon appears for resetting the value. After activating this icon the *Save* icon must be clicked to persist the changes.

Later on, we will describe the different parameter groups. Each of them is represented by an entry in the configuration menu. If you use a German server installation and encounter problems understanding the English terms used in this manual, we suggest to create and use an administrator with English language (the *sys* role is required in order to enter the administration).

Hint: The parameter definition and their groups are defined in *properties.xml*. This file should not be changed!

3.1.2 Location of configuration files

The configuration file (or the sequence of configuration files - see section below) has to be stated as environment variable *EP_CONFFILES* of *@enterprise* during the startup. The

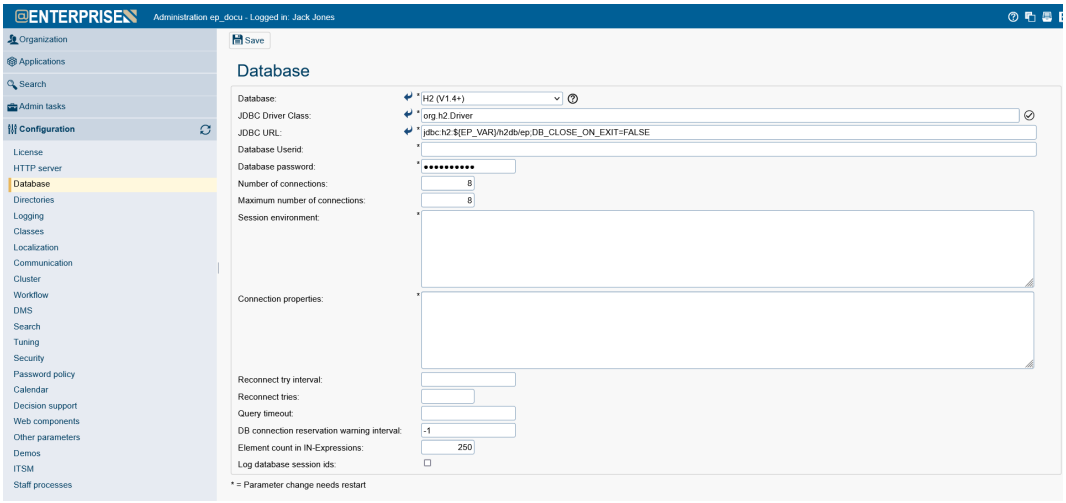

Figure 3.1: ***@enterprise* Configuration**

default mechanisms from the epstart.bat and epstart.sh locates it in the *conf/ep.conf* file below the *var* area.

- **Windows:** set environment variable *EP_CONFFILES* before calling epstart.bat, e.g.

```
set "EP_VAR=C:\wpwf\var"
set "EP_CONFFILES=%EP_VAR%\conf\myep.conf,%EP_VAR%\conf\commonep.conf"
```

- **Windows service:** set environment variable *EP_CONFFILES* in the *service\..\..\var\service\environment.bat* file before calling *service\epservice.bat* file for the procrun framework - or if you need to modify that file anyway set the variable in that file directly

- **Linux:** set environment variable *EP_CONFFILES* before calling epstart.sh,

```
export EP_VAR=/opt/epwf/var
export EP_CONFFILES=$EP_VAR/conf/myep.conf,$EP_VAR/conf/commonep.conf
```

- **Linux daemon:** as *Environment* line in the systemd unit description file

- **Application server:** as value of the context parameter in the WEB-INF/web.xml file **TODO michi**

Hint: up to @enterprise version 10, configuration files were given as command line arguments to the java command. This is no longer supported.

3.1.3 Multiple configuration files

@*enterprise* offers the possibility to specify a *sequence* of several configuration files, all filenames must have a `'.conf'` suffix.

If a parameter is available in more than one configuration file, the first appearance will be considered, additional occurrences in files later in the sequence are ignored. Changes in the values of the parameters are always carried out in the file with the first occurrence. If a new parameter (one that did not occur in any of the files) needs to be written, then it is written to the last file in the sequence.
This allows for a separation of parameters according to arbitrary criteria; e.g.:

- Semi-frozen configuration: use two files with different permissions/attributes: one file `avw_changeable.conf` which is readable and writable in the file system and another one `avw_frozen.conf` which is only readable. So all the parameters appearing in avw_frozen.conf will be locked.
- Clustered installation: use one file to define basic common parameters and one file (per node) to specify cluster node specific parameters.
- Similar installations: put all the parameters, which values in a production installation differ from the values of a test or staging installation into one file. Put all other parameters in a common one.

A sequence of files is specified as a list of file names separated by a comma or alternatively by the platforms path separator (`;` in Windows and `:` in Linux).

A file name can be a `*.conf` file or a directory. If a directory is given, all its `*.conf` files are considered in (case-insensitive) order

```
set "EP_CONFFILES=%EP_VAR%/conf/ep.conf,%EP_VAR%/conf/myep.conf,%EP_VAR%/my_conf"
```

In this example *ep.conf* and *myep.conf* are files, *my_conf* is a folder containing several configuration files.

3.1.4 Duration data type

The duration data type is used for all parameters which represent some time span. Times should be entered in the following format *[dD] [hH] [mM] [s [.f] S]*. For example:

- *1d 3h 2m 1s.* represents a time span of 1 day, 3 hours, 2 minutes and 1 second.
- *11h 34m 0.2s.* represents a time span of 11 hours, 34 minutes and 200 milliseconds.

Please note that if entered times are shorter than 1 millisecond, they will be saved as 0. Generally times are rounded to the nearest millisecond, which means that *2.0015s = 2.002s*. If negative times are entered, these are automatically converted to the value -1, as this is often used to deactivate a function.

3.2 License

The first screen contains license information:

- **License key - avw.license:** Your license key. If you want to change your license key after you finished the setup, you can enter the new key here.

3.3 HTTP server

This screen contains the setup parameters of the embedded HTTP server. It is not displayed in application server installations (such as Tomcat).

- **Server IP port - httpd.port:** HTTP port on which the server runs.
- **IP address - http.ip-address:** The default-behavior of multiple network-interfaces: the HTTP-server runs on all interfaces. With this parameter you can restrict the interfaces by entering an ip-address, where the server should run.
- **Minimum number of threads - httpd.minthread:** Number of threads, which are started on startup.
- **Maximum number of threads - httpd.maxthreads:** Maximum number of threads, which will be used for HTTP requests.

  Hint: If Apple Safari is used in combination with SSL, it is recommended to set an adequate high number for *Minimum Number of Threads* and *Maximum Number of Threads*.
- **Server SSL Port - ssl.port:** Port of the HTTPS server.
- **SSL IP address - ssl.ip-address:** Analog to parameter *IP address*, but for SSL port.
- **Client certificates for HTTPS - sl_requireclientcertificate:** This parameter determines how a secure SSL connection can be established by a client. There are three possibilities:
  - **Are not requested:** If this option is selected, SSL connections are established in any case.
  - **Are required:** If this option is selected, SSL connections are established only if the client has a valid certificate for authorization.
  - **Are requested:** If this option is selected, the establishment of SSL connections depends on the content of the response: if the response contains a valid client certificate the SSL connection is established automatically; if the response contains no valid client certificate a login mask will be displayed to the user and after a successful login the SSL connection will be established.
- **Administrative IP port - httpd.admin.port:** This port is used for administrating @*enterprise* - a admin session will be created which is necessary to execute administration functions. If no port is defined, the current user is already logged in and if he want to change to administration, he will be requested to log-in again (for getting admin session).
- **Administrative IP address - httpd.admin.ip-address:** Define an own IP address for administration. The behaviour is analog to parameter *Administrative IP port*.
- **Use SSL for administration - httpd.admin.usessl:** If activated, the *Administrative IP port* is a SSL port. If no port is defined, the *Server SSL Port* is used.

- **Allowed hosts or networks for administration - ep.adminshell.allowedips:** A list of hosts and networks can be specified. These hosts can access the administration of HTTP server. The syntax of this field is described below in section Defining Allowed and Denied Hosts or Networks.

- **Exclude SSL ciphersuites - httpd.jetty.sslconnector.excludeciphersuites:** Vulnerable SSL cipher suites can be excluded from use in HTTPS with following line:

```
httpd.jetty.sslconnector.excludeciphersuites=
TLS_RSA_WITH_AES_128_CBC_SHA,
\r\nTLS_RSA_WITH_AES_256_CBC_SHA,
\r\nTLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
\r\nTLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
\r\nTLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
\r\nTLS_ECDH_RSA_WITH_AES_256_CBC_SHA,
\r\nTLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
\r\nTLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
\r\nTLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
\r\nTLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
\r\nTLS_DHE_RSA_WITH_AES_128_CBC_SHA,
\r\nTLS_DHE_RSA_WITH_AES_256_CBC_SHA,
\r\nTLS_DHE_DSS_WITH_AES_128_CBC_SHA,
\r\nTLS_DHE_DSS_WITH_AES_256_CBC_SHA,
\r\nSSL_RSA_WITH_3DES_EDE_CBC_SHA,
\r\nTLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,
\r\nTLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,
\r\nTLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,
\r\nTLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
\r\nSSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
\r\nSSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA,
\r\nSSL_RSA_WITH_DES_CBC_SHA,
\r\nSSL_DHE_RSA_WITH_DES_CBC_SHA,
\r\nSSL_DHE_DSS_WITH_DES_CBC_SHA,
\r\nSSL_RSA_EXPORT_WITH_RC4_40_MD5,
\r\nSSL_RSA_EXPORT_WITH_DES40_CBC_SHA,
\r\nSSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA,
\r\nSSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA,
\r\nSSL_RSA_WITH_NULL_MD5,
\r\nSSL_RSA_WITH_NULL_SHA,
\r\nTLS_ECDH_ECDSA_WITH_NULL_SHA,
\r\nTLS_ECDH_RSA_WITH_NULL_SHA,
\r\nTLS_ECDHE_ECDSA_WITH_NULL_SHA,
\r\nTLS_ECDHE_RSA_WITH_NULL_SHA,
\r\nSSL_DH_anon_WITH_RC4_128_MD5,
\r\nTLS_DH_anon_WITH_AES_128_CBC_SHA,
\r\nTLS_DH_anon_WITH_AES_256_CBC_SHA,
\r\nSSL_DH_anon_WITH_3DES_EDE_CBC_SHA,
\r\nSSL_DH_anon_WITH_DES_CBC_SHA,
\r\nTLS_ECDH_anon_WITH_RC4_128_SHA,
\r\nTLS_ECDH_anon_WITH_AES_128_CBC_SHA,
\r\nTLS_ECDH_anon_WITH_AES_256_CBC_SHA,
\r\nTLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,
\r\nSSL_DH_anon_EXPORT_WITH_RC4_40_MD5,
```

```
\r\nSSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA,
\r\nTLS_ECDH_anon_WITH_NULL_SHA,
\r\nTLS_KRB5_WITH_3DES_EDE_CBC_SHA,
\r\nTLS_KRB5_WITH_3DES_EDE_CBC_MD5,
\r\nTLS_KRB5_WITH_DES_CBC_SHA,
\r\nTLS_KRB5_WITH_DES_CBC_MD5,
\r\nTLS_KRB5_EXPORT_WITH_RC4_40_SHA,
\r\nTLS_KRB5_EXPORT_WITH_RC4_40_MD5,
\r\nTLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA,
\r\nTLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5
```

The cipher suite used by a client can be seen via the URL `...servlet.method/com.groiss.cluster.ClusterSupport.getClientInfo`. Dealing with sporadic SSL-Handshake problems is greatly eased by setting the `javax.net.debug system` property in the Java command line, eg.:

```
-Djavax.net.debug=ssl:defaultctx:sslctx:handshake:verbose
```

This generates considerable amounts of log data, usage is only advisable when client connection issues via HTTS arise. An on-line assessment of your SSL parameters can be obtained at https://www.ssllabs.com/ssldb/index.html

- **Exclude SSL Protocols - httpd.jetty.sslconnector.excludeprotocols:** Protocols for Jetty SSL connectors can be specifically excluded. Please note that the default protocols being used depend on the specific Java version and release being used.
- **Context path - avw.contextpath:** This parameter defines the context path of *@enterprise*.

Hint: The default context path is "/wf." In standalone installations with embedded Jetty, you can designate a deployment in the root context of Jetty by using "/" or "ROOT". In installations within a servlet container, this property is irrelevant. Utilize the container-specific deployment mechanisms to define the context root. In Tomcat, deploying under the ROOT context is achievable by renaming the ep*.war file to ROOT.war.

3.4 Network access

Configuration of the network access control policies.

- **Allowed hosts or networks - httpd.hosts.allow:** Analogous to *Allowed hosts or networks for administration*, but only for non administration session. If *Allowed hosts or networks for administration* is empty and this host/network matches, it is possible to enter the administration (session).
- **Denied hosts or networks - httpd.hosts.deny:** Analogous to above, but only for non administration session.

- **Access control - urls.allowed:** We provide a mechanism which allows to grant or deny access to method-URLs based on a combination of IP-addresses and rights. The syntax of access rules and their semantics is described below in section Access Control.

3.4.1 Defining Allowed and Denied Hosts or Networks

To restrict access to the HTTP server to selected hosts or address ranges you can declare an *allow* and a *deny* list. The evaluation is as follows: If the allow-list is empty, access is allowed from every host except the ones in the deny-list. If the allow-list is not empty, access is allowed from the hosts and networks in the allow list minus the hosts (and networks) in the deny list.
Both lists contain pairs of IP-addresses and net-mask separated by spaces, commas or newlines. Both IPV4 and IPV6 addresses are permissible. A net-mask should be given in the CIDR style in form of an integer specifying the number of bits of the network-part. For IPV4 addresses, the traditional dotted notation is also permitted. An optional "P" before the address designates a proxy-address. Entries starting with # are ignored. See the following example:

```
10.205.112.0/255.255.255.0
P10.205.113.0/255.255.255.0
10.205.224.0/24
2001:0db8:0010::/48
```

This entries in the allow-list means, access from the networks `10.205.112.*`, (proxy-address) *10.205.113.0*,`10.205.224.*` and `2001:0db8:0010:*` is allowed. When entering IPV6 addresses directly in the config-file, bear in mind that each colon (`:`) must be escaped by preceding it with a backslash.
The following list used for the allow-list causes that access from hosts 10.205.112.4, 10.205.224.8 and 2001:DB8:0010:0:8:800:200C:417A is allowed.

```
10.205.112.4/32
10.205.224.8/255.255.255.255
2001:DB8:0010:0:8:800:200C:417A/128
```

3.4.2 Access Control

The access control mechanism affects the Dispatcher which serves URLs targeting java methods. Rules can be specified which restrict access to certain URLs based on a combination of IP-address and *@enterprise* rights.[1]

Configuration

The access control property consists of a comma-separated list of rules. Rules starting with # are ignored. Each rule combines an IP-specifier, an URL-prefix and a set of rights separated by spaces. Each of the components can be a wild card in the form of an asterisk.
Accordingly, the syntax of the ruleset is:

[1] It is no longer necessary to add the `com.groiss.avw.contrib.URLChecker` class as a service in the "services" field of the "classes" section.

```
{ ( ["P"] ip-specifier | "*" ) SPACE (url-prefix | "*") SPACE (
   "*" | "DENY" | ( right { SPACE right }* ) COMMA }*
```

The optional "P" before the ip-specifier designates a proxy-address.
Without specifying the "P", the remote address is the leftmost entry in "X-Forwarded-For" header field (if it exists), and the HttpRequest.getRemoteAddress(), if the header field does not exist.
When "P" is specified, the match is performed just against the HttpRequest.getRemoteAddress() without taking into account any "X-Forwarded-For" header.
The IP-specifier consists of an ip-address and a net-mask separated by a "/". Both IPV4 and IPV6 addresses are permissible. A net-mask should be given in the CIDR style in form of an integer specifying the number of bits of the network-part. For IPV4 addresses, the traditional dotted notation is also permitted. It can be used to specify a single host or a subnet in the following ways

`10.205.112.22/255.255.255.255`	designates the single host 10.205.112.22
`10.205.224.22/32`	designates the single host 10.205.224.22
`P10.205.112.23/255.255.255.255`	designates the proxy host 10.205.112.23
`10.205.112.0/255.255.255.0`	designates all hosts in the subnet 10.205.112.*
`10.205.224.0/24`	designates all hosts in the subnet 10.205.224.*
`10.0.0.0/255.0.0.0`	designates all hosts in subnet 10.*.*.*
`11.0.0.0/8`	designates all hosts in subnet 11.*.*.*
`2001:0db8:0010::/48`	designates all hosts in subnet 2001:0db8:0010:*
`::ffff:0a0a:0a0a/128`	designates a single IPV4 hosts 10.10.10.10
`*`	this wildcard designates all hosts

Technically, the IP-address of a requester matches an IP-specifier when the network prefix denoted by the netmask matches.
An URL-prefix consists of the first characters of a fully qualified method name (package, class, method) for the Dispatcher servlet. The URL-prefixes are case sensitive. There are two special URL-prefixes `webdav` and `wopi` for designating the URL-spaces in the WebDAVHandler and the WOPIHandler servlets.

`com.groiss`	designates all calls to methods in classes in packages located in `com.groiss` or below
`com.groiss.org.PasswdAuth`	designates all calls to methods in the class `com.groiss.org.PasswdAuth`
`*`	this wildcard designates all methods regardless of origin

The set of rights is a space separated list of IDs of *@enterprise* rights. The right IDs are case sensitive.

`set_agent`	designates all users who have the right `set_agent`
`admin stat`	designates all users who have the right `admin` and / or the right `stat`
`*`	wildcard designating that rights are not needed
`DENY`	special dummy right id, can be used to deny access

Examples for Rules

The following examples show how those three designations can be combined to form a rule:

`127.0.0.0/8 * *`
Access from local host subnet is not restricted.

`10.205.112.26/32 * DENY`
Access from `10.205.112.26` is not allowed.

`10.205.112.0/24 com.groiss.org.PasswdAuth *`
Login of hosts from subnet 10.205.112.0 is allowed.

`10.205.112.0/24 * internal`
All operations of hosts from this subnet are allowed if users have the right `internal`.

`* com.groiss DENY`
Access to `com.groiss.**` classes and methods is denied to every host.

`* com.my.appl admin customer`
Access to `com.my.appl.*` classes and methods is allowed if users have the right `admin` or `customer`.

`* * DENY`
Deny everything from everywhere.

Semantics

The validation of a list of rules in the *Access Control* property is as follows:
If the property is empty, nothing is filtered.
Otherwise all rules are checked in the order they are defined until a rule matches according to IP-specifier and URL-prefix. For a matching rule, the validation depends on the set of rights of the rule. We distinguish two cases:

- **Existing Session** (user already logged in):
  The intersection of the rights of the user and the rights given in the rule is computed. If the intersection is empty, access is denied (an exception is thrown), else the rule succeeds and access is granted.
- **No Session** (user not yet logged in):
  If the set of rights of the rule consists of a single DENY element, then access is denied (an exception is thrown), else the rule succeeds and access is granted.

If no rule at all matched, access is granted. This can be avoided if the last rule is "`* * DENY`".

Other Operational Considerations

Access Control gets reconfigured if the configuration is changed. This is also logged at log level 1 to allow one to find incorrect rules. Normal operations of *Access Control* are logged at log level 3.
Access Control is not automatically aware of additional rights given to a user or role or to the revocation of rights from them. In order to know about the constellation, the affected users must log out and log in again or the configuration must be saved (thereby reconfiguring *Access Control*). Caching of user rights in the *Access Control* mechanism is logged at log level 2.

3.5 Database

We suggest to use the help function (the question mark next to *Database*) to fill the Database, JDBC Driver Class, and JDBC URL fields with valid values for a selectable database.

- **Database - database:** The database; you can select ORACLE, DB2, MS SQL-Server, or Derby.
- **JDBC Driver Class - database.driver.class:** Java-Class, that contains the driver. See the table on page 40 for a list of driver classes.
- **JDBC URL - database.url:** URL for the database. The syntax of this string depends on the JDBC driver used. See the examples on page 40 or consult the documentation of the driver.
- **Database Userid - database.user:** The ID of the user with whom you want to connect to the database.
- **Database password - database.password:** Password for the database user with the ID that you entered above.
- **Number of connections - database.connections:** Default number of database connections.
- **Maximum number of connections - database.connections.max:** The maximum number of database connections that can be created.
- **Session environment - database.session.env:** You can specify SQL-commands, which are executed for each connection after connecting, for example: `set TEXTSIZE 1000000`
- **Connection properties - database.connection.properties:** You can specify e.g. SSL properties to establish a secure connection to database. The value of this property is a list of property declarations separated by `\r\n`. Note that the = sign must be escaped by \ when editing directly in *ep.conf*.

  `e.g. database.connection.properties=my.prop\=a.value\r\nyour.prop\=another.value`
- **Reconnect try interval - database.waitFor.seconds:** Interval for reconnect tries to the database. Duration data type parameter.

- **Reconnect tries - database.waitFor.count:** Number of reconnect tries.
- **Query timeout - database.query.timeout:** Interval after which a query times out. Duration data type parameter.
- **DB connection reservation warning interval - database.connection.busy.warning.secs:**

  Long-lasting reservations of DB connections can be logged and also monitored via the Server Monitor (Aged DB connections). Information includes thread-name, timestamp and stack trace at moment of reservation. Monitoring information in the log file will occur in 2 minute intervals. Each long-lasting reservation is logged not more than once. Following values can be defined:

  - -1 : do not monitor (default, behavior like before)
  - >=0 : do monitor; log /report all connections being reserved longer than the specified time interval. Duration data type parameter.
- **Element count in IN-Expressions - ep.inlists.splitsize:** @*enterprise* uses SQL "IN-lists" - SQL expressions of the form WHERE att IN (val1, val2, ..., valn) as one form of optimizing database access.

  The API provides the `com.groiss.store.BulkQuery` class as a convenient means to utilize this fast kind of access.

  Since database systems usually put restrictions on the textual length of SQL-statements and also on the number of elements in such IN-lists, @*enterprise* splits queries with long IN-expressions into several queries. This configuration parameter can be used to control the maximum number of elements of an IN-expression.

  The default value is 250 elements. Increasing the parameter leads to fewer partial queries and fewer roundtrips to the database, but also to longer statements and IN-lists with the possibility to hit the limits imposed by your DBMS.
- **Log database session ids - database.logdbsessionid:** Check this parameter to log session ids of database (Oracle, SQLServer). To include database session ids in the log, it is necessary, that the database user *SYS* executes the following grant:

  ```
  grant select on v_$session to ep;
  ```

Table 3.1 shows the recommended drivers for the databases, their class names and JDBC URLs (you can directly view and use this table in @*enterprise* by clicking on the help link next to *Database*).

3.6 Directories

Here you can define some directories that @*enterprise* will use. The *Directory of Form Classes* and *Directory for Temporary Files* must exist.

- **Directory of form classes - avw.formclassdir:** Directory, where the system writes the form classes.
- **Directory for temporary files - Httpd.tempDir:** Directory for temporary files.

DBMS	Driver Vendor	Driver Kind	Class and URL
DB2 UDB	IBM	Data Server Driver	com.ibm.db2.jcc.DB2Driver jdbc:db2://'host':50000/'dbname'
Derby	Apache	Embedded	org.apache.derby.jdbc.EmbeddedDriver jdbc:derby:ep;create=true
H2	H2 Community	Embedded	org.h2.Driver jdbc:h2:./'path'/'dbname'; DB_CLOSE_ON_EXIT=FALSE
MS-SQLServer (V2005+)	Microsoft	V3.0+	com.microsoft.sqlserver.jdbc.SQLServerDriver jdbc:sqlserver://'host':1433; encrypt=false;database='dbname'
MySQL (V5.0, experimental)	MySQL	Connector/J (3.1)	com.mysql.jdbc.Driver jdbc:mysql://'host':'port'/'dbname'
Oracle LOBs	Oracle	Thin (V10g+)	oracle.jdbc.OracleDriver jdbc:oracle:thin:@'host':1521:'SID'
Oracle LOBs	Oracle	OCI	oracle.jdbc.OracleDriver jdbc:oracle:oci:@'TNSNAME'
PostgreSQL (V8.1+)	PostgreSQL	Native	org.postgresql.Driver jdbc:postgresql://'host':'port'/'database'
MS-SQLServer (-V2000,deprecated)	Microsoft	V3.0+	com.microsoft.sqlserver.jdbc.SQLServerDriver jdbc:sqlserver://'host':1433;database='dbname'
Oracle LONGs (deprecated)	Oracle	Thin	oracle.jdbc.OracleDriver jdbc:oracle:thin:@'host':1521:'SID'
Oracle LONGs (deprecated)	Oracle	OCI	oracle.jdbc.OracleDriver jdbc:oracle:oci:@'TNSNAME'

Table 3.1: JDBC-Drivers

Hint: Please note that the new configuration values for temporary and forms directory will not be used before server restart.

3.7 Logging

- **Log file - logger.logfile:** Name (path) of file, where *@enterprise* writes log information. If file not exists, a new one will be created. If the log file extension is .zip or .gz, the log file(s) will be compressed automatically depending on parameter *Restart log* or *Max. filesize*.
- **Restart log - logger.restart.logfile:** Here you can define how often the log file should be initialized - daily (at midnight) or at startup only.
- **Number of logs - logger.keep.logfile:** The number of stored log files. If the log file extension is .zip or .gz, the log file(s) will be compressed automatically.
- **Error file - logger.errorfile:** This file is a centralized collection of errors. Errors will also appear in the general log file. If the log file extension is .zip or .gz, the log file(s) will be compressed automatically depending on parameter *Restart error log* or *Max. filesize*.
- **Restart error log - logger.restart.errorfile:** see *Restart log*
- **Number of error logs - logger.keep.errorfile:** see *Number of logs*
- **System loglevel - ep.logger.level:** This setting is used for log actions of *@enterprise* (applications) only. One of the following levels can be selected:

  Inherit If selected, level of parameter *root.logger.level* in section *Other parameters* is taken.

  ERROR Errors are logged only.

  WARN In addition to level *ERROR* warnings are logged.

  INFO HTTP requests are logged (time stamp, user, IP-address, and URL).

  DEBUG SQL-statements and process-oriented logging.

  TRACE The full HTTP-headers, parameters of prepared statements and other information for debugging purposes.

  Don't use the options DEBUG or TRACE in production for extended periods of time, because that generates a lot of log data.

- **Store loglevel - store.logger.level:** Analog to *System loglevel*, but is used for log actions of *@enterprise* Store (package *com.groiss.store*) only. You can select the entry *Inherit* to use the selected level of *System loglevel*.
- **Servlet loglevel - httpd.logger.level:** Analog to *System loglevel*, but is used for log actions of *@enterprise* servlet methods (package *com.groiss.servlet* and Java Melody) only. You can select the entry *Inherit* to use the selected level of *System loglevel*.

- **Log on console - logger.logOnConsole:** The log information is written to the standard output stream.
- **Custom log levels - logger.custom.level:** @*enterprise* has different loggers which can be customized here with a list separated by semicolons. It is possible to increase or decrease the trace level for a logger like in following example:

```
com.groiss.servlet.impl.Dispatcher=ERROR;
com.groiss.store.impl.StoreEJB=DEBUG;
```

  Please note that all other loggers use the default settings. We also provide special logger names which can be used to change the standard behavior of some components as summarized in the following table:

loggername=LEVEL	Effect
mail=DEBUG	logs mail protocol traffic (*SMTP, IMAP, POP3*)
cometd=DEBUG	adds additional cometd message logging on client side
dojo=DEBUG	adds additional dojo related logging on client side. Avoids loading of layer files and tries to load each module separately which facilitates easier JavaScript debugging

- **Trace levels for threads - logger.thread.levels:** To allow for finer logging in background threads, thread-specific log levels can be defined via a list of entries of the form *threadname=loglevel*, like e.g.

```
ClientNotificationDispatcher=DEBUG;
EventDispatcher=TRACE;
```

- **Application loggers - logger.application.packages:** Define a comma separated list of application loggers as package/class/logger name, e.g. *com.groiss.itsm* means that all logging actions of this package are using the *System loglevel*.
- **Length of tail - logger.tail.length:** This parameter defines how much rows are displayed by default at the end of a log file via GUI (see

  Admin tasks → Server → Logging).
- **Max. filesize - logger.max.filesize:** This option can be specified in bytes, kilobytes, megabytes or gigabytes by suffixing a numeric value with KB, MB and respectively GB. For example, 5000000, 5000KB, 5MB and 2GB are all valid values.
- **Number of configuration backups - keep.conffiles:** This parameter defines how many backups of the configuration files (ep.conf and self-defined configuration files) should be kept. If saving the configuration via GUI, a backup file will be created in the appropriate folder where the configuration files exist.

3.8 Classes

- **Authorization Class - HttpdAuth.class:** *@enterprise* allows the usage of different authorization mechanisms. The Java class used is specified here. The default class (part of the distribution) is `com.groiss.org.PasswdAuth`.
  A special class is `com.groiss.ldap.LDAPPasswdAuth` which allows to authenticate against a LDAP server. The password check at login is delegated to such a server. After setting this class and reloading the configuration navigation tree, a new section *User authorization via LDAP* will appear (more details are available in section User authorization via LDAP).
- **Configuration Amender Class - ep.configuration.amender.class:** The configuration can also be amended via a Java class. This class must implement the interface `com.groiss.component.ConfigurationAmender` and it must be resolvable via the local class path. Use cases for such a class would be fetching of configuration data or secrets from external sources (environment variable, vaults . . .). A toy example for such a class can be found in the demo package (`com.groiss.demo.DemoConfigurationAmender`).
- **Settings Class - settings.class:** A class defining some global settings can be defined here. For details see the *@enterprise* Programming Guide.
- **Notification Provider Class - avw.notification_provider_class:** The class for the notification mechanism, must implement the interface `com.dec.avw.notification.NotificationKit`. This mechanism allows to notify RMI clients in an asynchronous manner about changes in worklists.
- **Archiving Class - avw.archiveclass:** The class used for archiving process instances, must implement the interface `com.groiss.wf.ProcessArchiver`. If archiving should be prevented, configure `com.groiss.wf.NoArchiver` as archiving class!
- **Error-Formatter Class - avw.error.formatter:** You can write an error formatter class that will be used to display errors. The class must implement the `com.groiss.gui.ErrorFormatter` interface.
- **Services - ep.services.extensions:** Here you can add your own services.
- **Form class package - ep.form.class.package:** This parameter allows to define the default form class package for new created form types (existing form types keep their previous package name).

3.9 Localization

- **List of locales - Locale.list:** Here you can define a comma-separated list of locales that will be used by the server. If you don't define anything here, the server will use the following default locales: `en_GB`, `en_US`, `de_DE`, `de_AT`, and `de_CH`.
- **Language - Locale.language:** Defines the language for the user interface. Language is defined in ISO language code, for example `de` for German.

- **Country - Locale.country:** ISO country code, for example `AT` for Austria.
- **Variant - Locale.variant:** A default variant to use. You can define free variants in the list of locales (e.g., regions, companies, etc.).
- **Server timezone - avw.timezone:** This parameter allow to enforce a specific timezone for all users. If nothing is selected, the default timezone of the server (operating system) is used, otherwise the selected one.
- **Decimal format - avw.decimal.format:** Define a decimal format as described in JAVA APIDoc.
- **Decimal separator - avw.decimal.separator:** Set the separator for floating-point numbers (default is .)
- **Decimal grouping separator - avw.decimal.grouping.separator:** A separator for e.g. thousand delimiter can be defined here (see Java APIDoc for more details).
- **Date format - DateFormat:** Format mask for date input and output. See section Date and time formats below for a description of the possible values.
- **Time format - TimeFormat:** Format mask for time input. See the section Date and time formats below for a description of the possible values.
- **Default unit for displaying time intervals - calutil.defaultunit:** Default-Unit in seconds, minutes, hours, days and weeks.
- **Max. table length - table.maxsize:** Specify a natural number. For tables of size greater than this number the user is asked before the table is shown.
- **Items per page - table.pagesize:** This defines the maximum number of entries in tables when paging is enabled. This parameter is also used for DOJO object selects.
- **Max. paging table length - table.paging.maxsize:** For paged tables of size greater than this number the user is asked before the table is shown or the search function in toolbar must be used.
- **Use browser language - locale.from.browser:** If this option is set, the system uses the language settings of the browser instead of the settings in the user table of *@enterprise*.
- **Always use server-timezone - use.server.timezone:** If this checkbox is set, the determined timezone of client Browser will be ignored and either the timezone set by the user (on user mask or user settings mask) or the *Server timezone* will be used.
- **Select list search option - selectlist.search:** The search option for searching in a select list:
  - Starts with: at the begin of a string
  - Substring: within a string
- **Enable Wiki link syntax - ep.wikilinks.enable:** If this checkbox is activated, links can be entered in the description of an ActivityInstance in wiki syntax: [[link | text]] or [[link]]

Symbol	Meaning	Presentation	Example
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July & 07
d	day in month	(Number)	10
h	hour in am/pm (1 12)	(Number)	12
H	hour in day (0 23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday
D	day in year	(Number)	189
F	day of week in month	(Number)	2 (2nd Wed in July)
w	week in year	(Number)	27
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day (1 24)	(Number)	24
K	hour in am/pm (0 11)	(Number)	0
z	time zone	(Text)	Pacific Standard Time
’	escape for text	(Delimiter)	
”	single quote	(Literal)	’

Table 3.2: Values for Date and Time Format Masks

- **Default tab in process details:** Define a tab id as default when opening a process detail window. Default value is “admin.history”. See the System Administration Manual, section Processes, for details (field “Detail tabs” in process definition mask).
- **Show toolbar in process-details popup:** When the process details are opened in a separate window, this checkbox enables a toolbar showing the possible actions for this process instance. For example, if you use the standard search for searching a process instance that is in your worklist, you will see the worklist actions in the search result details of this process.

3.9.1 Date and time formats

Table 3.2 shows possible values for the date and time format masks.
The count of pattern letters determine the format.
(Text): 4 or more pattern letters–use full form, < 4–use short or abbreviated form if one exists.
(Number): the minimum number of digits. Shorter numbers are zero-padded to this amount. Year is handled specially; that is, if the count of ’y’ is 2, the year will be truncated to 2 digits.
(Text & Number): 3 or more–use text, less than 3–use number.
Any characters in the pattern that are not in the ranges of [’a’..’z’] and [’A’..’Z’] will be treated as quoted text. For instance, characters like ’:’, ’.’, ’ ’, ’#’ and ’@’ will appear in the resulting time text even if they are not embraced within single quotes.

Date and time formats can be set to be empty by explicitly entering " " or by using the keyboard combinations ALT+0160 or ALT+255 in the parameter field. Please note that using a single "ordinary" space character, an empty string or deleting the format will not

work, because such values would not outlive a server restart or a configuration reload.
Hint: To avoid the display of the time picker components, add the following to your style definitions:

```
.scDateField .scTimePicker {
   display: none;
}
```

3.10 Communication

- **SMTP host - mail.smtphost:** Server for outgoing emails (host name or IP address). It is also possible to define the smtp port in this field which must be separated by a colon, i.e. <smtp_host>:<smtp_port>.
- **Mail sender - mail.sender:** The mail address that will appear in the *from* field of mails that the system sends.
- **SMTP Username - ep.mail.smtp.username:** The user name for SMTP server (SMTP host).
- **SMTP Password - ep.mail.smtp.password:** The password of SMTP user.
- **Type of SMTP communication - ep.mail.smtp.communicationtype:** This setting is used by all communication possibilities in @*enterprise* for sending mails. One of the following communication types can be defined here:
  - *Unencrypted:* The content of the mail will be transferred without encryption. This is the standard communication type.
  - *STARTTLS:* This is an extension to plain text communication protocols, which offers a way to upgrade a plain text connection to an encrypted connection instead of using a separate port for encrypted communication.
  - *Encrypted:* The mail will be SSL–encrypted. The validity of the mail server certificate will not be checked.
  - *Trusted (with certificate):* To assure a secure transmission the mail server has to authenticate itself adverse @*enterprise*. This is achieved by checking the mail server certificate. To add a new certificate for a mail server it has to be added to the key store of @*enterprise*.
- **Administrator email address - avw.adminemail:** One ore more email addresses of the system administrator separated by comma. Beside this field the check function allows to test the SMTP settings by sending an email to administrator's email address. If a timer doesn't catch an exception, @*enterprise* sends a mail to the system administrator and deactivates the timer.
- **Subject pattern - ep.mail.subjpattern:** An email subject consists of `<subject>` `(<pattern> <pid>)`. In this field only the <pattern> part could be entered which is needed for identification, e.g. the text *ID:*. If subject pattern is entered, the email will be assigned to an existing process with given <pid> (if available). After this attempt the given *Action* of mailbox is executed.

- **Non trustworthy senders - ep.mail.junk:** The mails of all mailboxes will not be handled for given email addresses or a pattern of addresses. Separators are new lines. Examples:
  - `*groiss*` - If email address contains string "groiss", email will not be handled by mailbox
  - `*groiss.com` - If email address contains string "groiss.com" at the end, email will not be handled by mailbox
  - `max.muster@*` - If email address contains string "max.muster@" at the beginning, email will not be handled by mailbox
  - `max.muster@groiss.com` - If email address contains exactly the string "max.muster@groiss.com", email will not be handled by mailbox
- **Default action for sending mails - ep.mail.queue.usage:** Here you can define the default action for sending emails. Following values are available:
  - `Send over mail queue`: This option allows to send mails by using mail queue. If an error occurs, the mail will be stored in mail queue until *MailQueueTimer* is executed (see System Administration Guide - section *Timers* for more details).
  - `Put in mail queue`: If this option is selected, mails are stored in mail queue without sending attempt. The mails are sent when *MailQueueTimer* is executed (see System Administration Guide - section *Timers* for more details).
  - `Send without mail queue`: If this option is selected, mails will be sent immediately without using the mail queue. This is the default setting.
- **Max. time for mail queue item - ep.mail.queue.maxtime.minutes:** The *MailQueueTimer* iterates over the mail queue and handle each entry which has a creation date. This creation date is used for this configuration parameter and if max. time is exceeded, the administrator will be informed and a appropriate status message will be written for this mail queue entry. The default value is 24 hours. Duration data type parameter.
- **SMTP default properties - ep.mail.smtp.defaultprops:** Define default properties for SMTP mail communication (see http://javamail.java.net/nonav/docs/api/). In particular the following properties are useful in dealing with network problems: *mail.smtp.connectiontimeout* and *mail.smtp.timeout*.

  Please note that the properties *mail.smtp.host* and *mail.smtp.port* cannot be overwritten by using this field, because the definition is done with *@enterprise* configuration parameter *SMTP host - mail.smtphost*!
- **IMAP default properties - ep.mail.imap.defaultprops:** Define default properties for IMAP mail communication (see http://javamail.java.net/nonav/docs/api/). In particular the following properties are useful in dealing with network problems: *mail.imap.connectiontimeout* and *mail.imap.timeout*.
- **POP3 default properties - ep.mail.pop3.defaultprops:** Define default properties for POP3 mail communication (see http://javamail.java.net/nonav/docs/api/). In particular the following properties are useful in dealing with network problems: *mail.pop3.connectiontimeout* and *mail.pop3.timeout*.

- **Enable Wf-XML - avw.wfxml.enabled:** Defines, if this server is Wf-XML enabled. Possible values are *Off*, *Active*, or *Passive*. For further details on how to set up and use Wf-XML, please take a look at the section

  Communication with other Systems → Wf-XML

  of the *@enterprise* Application Development Guide.

- **WfXML Org.-Unit - wfxml2.orgunit:** Default Wf-XML Organizational Unit.
- **WfXML User - wfxml2.user:** Default Wf-XML User.
- **WfXML Server - wfxml2.server:** Defines the default Wf-XML server.
- **WfXML access log for - wfxml2.log.objects:** Defines the objects, which will be logged. You can select between
  - ServiceRegistry
  - Factory
  - Instance
  - Activity
  - Observer.
- **Size of log - wfxml2.log.size:** Max. size of the log file.

3.11 Cluster

See section Adapting the *@enterprise* Configuration in the chapter about clusters for details about configuring clusters.

3.12 Workflow

- **Open form on process start - avw.start.with.form:** In the process start mask there is a checkbox where the user can decide to see the process form immediately after process start. Here you can define the default value of this checkbox.
- **Inherit Ids to subprocesses - avw.inherit.ids:** Don't create Ids for subprocesses - use the parent processes' Ids instead.
- **Enable application-spanning process definition - avw.procdef.appl_spanning:** If this option is set active (by default), it is possible to define processes with application-spanning elements (i.e. Forms, Tasks, Subprocesses and Roles as Agents). If this checkbox is not checked, elements of the current application are available for process definition only.
- **Allow automatic take - avw.autotake:** Allows users to take tasks automatically if they perform a function directly on an entry in the role-worklist or suspension worklist. This will only work if you add additional functions to the GUI of these worklists (e.g., the `finish` function). If the process-form of such a task is edited, the current editor is written in table *avw_currenteditor* and is visible in the process-instance history.

- **Show choice selection when single path - ep.choice.showsingle:** If this checkbox is not activated, no choice-mask is displayed anymore when one branch of a choice-object is active only. If activated, the choice-mask is displayed always.

3.13 DMS

- **Versioning - avw.dms.versioning_strategy:** *Not automatically* disables automatic version creation. *On agent change* creates a version if a different user edits the document (so, if the same user edits a document multiple times, no documents are created). *On every change* creates a version every time the document is edited - an exception is the adaption via WebDAV client (e.g. MS Word): if a document is opened and stored multiple times, only one version is created (= for each "session" only one version is created).
- **Propagate permission list - avw.dms.bequest_acl:** When this option is checked, the permission list of a DMS folder will be assigned to each object being added to that folder. Please note: if the permission list assignment of a folder is changed this has no effect on the permission list assignment of the objects already contained in that folder.
- **DMS Storage Class - IStore.class:** You can specify your own DMS storage class here. The class must implement the interface `com.groiss.dms.IStore`.
- **DMS Archiving Class - DMSArchiver.class:** Class for archiving documents, must implement the interface `com.groiss.dms.DMSArchiver`.
- **Standard table model / Table handler - avw.dms.standard_tablemodel:** A class can be specified, which is used for displaying the document tables. For further details please take a look into *@enterprise* Application Development Guide, section *Using the DMS API*.
- **WebDAV drive - webdav.drive:** The webdav drive can be specified with this parameter, which represents the root (the same letter like set in WebDrive properties). If the value *off* is entered, WebDrive will not be used anymore. You have to reconnect to the *@enterprise* Server after changing this parameter.
- **Open documents with - webdav.officeDocuments.application**:

  This setting is used to determine, how office-documents should be opened in the DMS. One of the following ways can be selected:

  - *Browser Default*: The documents are simply downloaded.
  - *Microsoft Office Plugin*: The documents are opened with the Microsoft Office plugin. More information can be found in section Edit Microsoft Office Documents via Browser.
  - *Office Online*: The Documents are opened with the online version of either Microsoft- or Libre Office. More information can be found in section Edit Office Documents via Office Online.

- **Extensions of documents supported by plug-in and Office Online - webdav.officeDocuments.openWithPlugin**:

  Holds a comma separated list of extensions (e.g. doc, docx) for which the Microsoft Office Plugin or Office Online should be used. More information can be found in sections Edit Microsoft Office Documents via Browser and Edit Office Documents via Office Online.

- **Office application - dms.officeOnline.officeSuite**:

  Is used by Office Online to determine, which office suite should be used to open office documents in the DMS. One of the following applications can be used:

  - *Microsoft Office Online*
  - *Libre Office Online*

  More information can be found in sections Edit Microsoft Office Documents via Browser and Edit Office Documents via Office Online.

- **URL of the discovery XML - dms.officeOnline.discoveryURL:** Is used by Office Online to get the discovery XML. More details about this XML can be found in section Edit Office Documents via Office Online.

- **Allow Co-Authoring in Office Online - dms.officeOnline.allowCoAuthoring:** Is used by Office Online to enable or disable simultaneous editing of office files. This only works in Microsoft Office Online.

- **Open text files via text editor - dms.use.texteditor:** This must be checked to activate usage of the text editor in the *@enterprise*.

- **Extensions of text editor supported files - dms.use.texteditor.extensions:** Holds a comma separated list of extensions for which the text editor should be used. Only for DMS documents which extension is contained in that list the editor will be used. Supported document types are: txt, asc, csv, etx, rtx, tsv, wml, wmls, xml, htc, css.

- **Use image-previewer - dms.use.image.preview:** If checked, the image-previewer of *@enterprise* is used for displaying images stored in DMS, otherwise the image is opened in an own Browser tab.

- **Maximum document size (in bytes) - avw.dms.max_doc_size:** You can define a maximum size for DMS documents here. *@enterprise* will not allow users to create documents that are bigger than this value. If you don't define a maximum size, there will be no size restriction for DMS documents. Anyway, also databases can limit the maximum size.

- **Character set for text files - avw.dms.textfile_charset:** Here you can enter the character set for text files, if the content of these files is not displayed correctly, e.g. the content of the file has ANSI charset, but the server charset is UTF-8 - for this purpose set the character set for text files to the value *CP1252* (if client is running under Windows only).

- **Full-text search - avw.dms.textsearch.state:** With the help of this parameter the state of the full–text search can be determined: There are three possible states:
  - **Switched off:** No full–text search is used at all.
  - **String search in form fields:** The database doesn't support full–text search. Therefore the required string can be searched in a table containing all string values of form fields.
  - **Activated:** The full–text search of the current database is used.
- **Check permissions on DMS folder content - dms.check.rights.on.list:** If activated, the view-right is checked when folder content is read.
- **Use Recycle Bin - ep.dms.use.recyclebin:** If this checkbox is activated, the recycle bin for DMS objects is used, i.e. if a DMS object is deleted, it will be moved to recycle bin first and will not be deleted. More information concerning this topic is available in the @*enterprise* user manual.
- **Show recently used documents - ep.dms.showrecentlyused:** If checked, document usage information is maintained to enable users to see the documents they used recently.
- **Show extensions - avw.dms.showextensions:** Show the document name extension, e.g., *.doc* or *.txt*.
- **Do not display hidden documents - avw.dms.hide_hidden_docs:** If this option is checked, users cannot see any hidden documents (beginning with a point in the filename) in the DMS.
- **Do not display folder Common - dms.hide.common:** With this parameters you can hide the *Common* folder in DMS.
- **Do not display user related folders - dms.hide.userfolder:** This parameter fades out the user folder and the folder of the substituted person
- **Sort table grouped by folders/documents - dms.grid.sort.grouped:** If this parameter is checked the elements of a DMS folder will be grouped into folders and non-folders and sorting (e.g. by name) will be performed within this groups (as it is known by Windows Explorer). This parameter works in smartclient only!
- **OpenOffice home - ep.openoffice.path:** The root path of OpenOffice can be entered here for replacement of Office templates (odt-files). More details about Office templates in @*enterprise* can be found in the *Application Development Guide*. Please note that a mixture of 32-bit and 64-bit version of JAVA and OpenOffice can lead to problems (e.g. converting mechanism from odt to another file-format cannot be used)!
- **Number of named OpenOffice pipes - ep.openoffice.threads:** Here you can enter the number of threads which are used for the piped connection with OpenOffice. The default value is 1.

- **Use named pipes to connect to OpenOffice - ep.openoffice.piped.connection:** If this checkbox is activated, a piped connection will be established with OpenOffice. This option is activated by default.
- **Used ports for OpenOffice connection - ep.openoffice.ports:** Alternative to named pipes connection a connection via ports (socket connection) can be used. For this purpose a port (or a comma separated list of ports) must be entered and the checkbox *Use named pipes to connect to OpenOffice* must be deactivated. The default port is 210.
- **Support conversion to 'Office Open XML' types (docx, xlsx, etc.) - ep.openoffice.support.office.open.xml:**

  Activate this checkbox, if the function *Generate document (generate_doc)* is used to create documents and if you are utilizing LibreOffice for this (OpenOffice does not support such types).
- **Generate Thumbnails - ep.dms.thumbnails.use:** If this checkbox is activated, thumbnails are generated for files in the DMS. To generate thumbnails for Open- and MS-Office documents, as well as Text files, an OpenOffice installation is needed. Furthermore, make sure that the properties "Used ports for OpenOffice connection" and, if named pies are used, "Number of named OpenOffice pipes" are set to a reasonable value for a multi-user environment, because every time a file is added to a folder by a user, a thumbnail is generated of it. Providing insufficient resources can cause long delays or, in the worst-case, multiple errors. This option is activated by default.
- **Files ignored in zip upload - ep.dms.zipupload.ignore:** Enter a comma separated list of path names which are ignored by DMS function *Zip upload*. Allowed are also * and ? as wildcards. The whole path is always compared (case-sensitive), i.e. if you want to hide directory .xx and its content, you have to enter *.xx**

3.13.1 Edit Microsoft Office Documents via Browser

Via WebDAV it is possible to open Microsoft Office Documents in read-write mode when clicking on a document link in the browser. Therefore the new *@enterprise* GUI will use the browser plug-in that will be automatically installed if Microsoft Office is installed on your computer. For using that feature two aspects must be configured:

1. activate the usage of the plug-in
2. configure user authentication

Plug-in activation: Therefore section DMS of the configuration provides the following parameters:

- **Open documents with - webdav.officeDocuments.application**:

  Microsoft Office Plugin should be selected to activate usage of the plug-in by *@enterprise*.

- **Extensions of documents supported by plug-in and Office Online - webdav.officeDocuments.openWithPlugin**:

  Holds a comma separated list of extensions (e.g. doc, docx) for which the Microsoft Office Plugin should be used. Only for DMS documents which extension is contained in that list the plug-in will be used - all other documents will be handled with the browser's default behavior.

Note: It may be the case that the plug-in is deactivated by the browser itself. In that case it must be activated within the browser application (e.g. via Add-Ons/Plugins in FireFox). If a Browser (e.g. Google Chrome) does not support the plug-in, the MS Office URI schemes are used to open the MS Office document in an editable way!

Authentication: For viewing/editing a DMS document we must determine the requesting user to check if he is permitted for that action. But the session cookie of the browser cannot be passed to the plug-in therefore we need alternative ways for authentication of the requesting user. The following parameters in section Security of the configuration are determined to control the various ways for authentication:

- **Allow Basic-Auth for WebDAV - avw.dms.allow_basicauth**: When activated the server will send an authentication request to the client if no user could be determined. The client will react by opening a dialog to enter the user's id and password that will then be sent to the server.

  Usage of Basic Authentication may be deactivated on your windows client but you may change the current behavior by setting registry entry HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WebClient\Parameters\BasicAuthLevel.

  Supported values are:

  - 0 - Basic authentication disabled
  - 1 - Basic authentication enabled for SSL shares only
  - 2 or greater - Basic authentication enabled for SSL shares and for non-SSL shares

  When setting value to 2 be aware that username/password will be transmitted in clear text when using a non-SSL connection.

  Note: Activating Basic-Auth in *@enterprise* is not only relevant for editing documents via the browser but for all kinds of WebDAV clients trying to connect to the DMS of *@enterprise*.

- **Use authentication token in DMS - ep.dms.useauthtoken**: When checking this parameter an authentication token will be passed as part of the URL to Microsoft Office. The server will be able to authenticate the user by this token.

- **WebDAV authentication class - webdav.auth.class**: Here you may specify an implementation of interface `com.groiss.servlet.WebDAVAuth` which will be used to

determine the requesting user by some data in the HTTP request (e.g. from a client certificate sent as part of that request).

As for Basic-Auth this setting will be used for all kinds of WebDAV clients.

3.13.2 Edit Office Documents via Office Online

With Office Online, it is possible to view office documents and edit them collaboratively, which means, that multiple users can edit a file simultaneously and all changes can be seen by each participant in the edit session. Therefore, *@enterprise* implements the Web Application Open Platform Interface (WOPI) protocol to communicate between *@enterprise* and the Office Online editor. Currently, following applications are supported:

- Microsoft Office Online
- Libre Office Online

To activate Office Online, parameters must be configured as follows:

- **Open documents with - webdav.officeDocuments.application:** Office Online has to be selected.
- **Office application - dms.officeOnline.officeSuite:** The correct Office application must be chosen.
- **URL of the discovery XML - dms.officeOnline.discoveryURL:** A correct URL to a discovery XML is required.

A vital part in any Office Online suite is the Discovery-XML. It provides information about the capabilities that Office Online applications expose, and how to invoke them. The URL to this file must be entered in the **URL of the discovery XML** - property. The form of the URL usually is `https://<server>/hosting/discovery`. Without this, Office Online will not work at all. The configuration of each office application is described below.

Microsoft Office Online (MSOO):

Microsoft Office Online was tested with the *WOPI Validation Application* of the Cloud Storage Partner Program. To use MSOO on premise, a Office Online Server needs to be set up. More information can be found here:
https://docs.microsoft.com/en-us/officeonlineserver/office-online-server.

Libre Office Online (LOOL):

Information about Libre Office Online can be retrieved on the official Homepage of Libre Office (**https://www.libreoffice.org/download/libreoffice-online/**). The LOOL integration was tested in the *Collabora Online Development Edition (CODE)*. The binaries, general information about the project and an instruction how to setup such a server can be retrieved here:
https://www.collaboraoffice.com/code/.

3.14 Search

- **Maximum table size on server (rows) - query.maxtable:** Maximum table size the server will handle. If the table size exceeds this value, the operation is canceled and an error message is produced.
- **Cache interval - monitoring.cacheinterval:** Specifies, how long a query result resides in cache. Duration data type parameter.
- **Maximum number of cached queries - monitoring.cachesize:** Number of queries in cache.
- **Maximum number of simultaneous queries - monitoring.maxparallel:** Number of threads, that concurrently compute query results.
- **Maximum number of startable queries - monitoring.maxqueue:** Length of queue of queries waiting for execution (waiting for a free thread).
- **Default process Id search type - avw.reporting.defaultIdSearch:** Here you can define the standard type for id search in *Process Search* - see user manual for further information.
- **Default subject search type - avw.reporting.defaultSubjectSearch:** The same as *Default process-id search type*, but for subject.
- **Process relations - avw.process.relations:** It is possible to define a relationship between process instances. The relation is defined as *ProcessRelation(ProcessInstance p1, ProcessInstance p2, String reltype)*. The relation can be maintained via API or with the task-function *addRelation*. The available relation types can be defined in the field *Process relations*. For each relation type a pair of id and name1/name2 is defined, names and id separated by whitespace (see syntax beneath). A comma, new line feed or carriage return separates the pairs. The id is stored in the database relation, the names are used in the user interface.

  Definition syntax:

  ```
  id [name1 [name2]]{sep id [name1 [name2]]}
  ```

  If name2 is missing, name1 is the default. If name1 is missing, the id is the default. In name1 and name2 it is possible to use %20 to use blanks in names (values). The names could be internationalized by adding @@@appid:key@@ (appid is the ID of the application; *ep* is @*enterprise* default).

  Examples:

  ```
  a
  b B
  c C%201 C2
  d @@@ep:process@@
  e @@@ep:forward@@ @@@ep:back@@
  ```

- **Process relation display - avw.process.relations.display:** A pattern which defines how the process relations in history will be displayed.
  The default pattern is @@@*process*@@ *id: subject*.
  There are several variables that can be used to customize how the relation will be displayed:
  - **id**: process ID
  - **ou**: process organization unit
  - **subject**: subject of the process instance
  - **process**: name of the process definition
- **Search case-insensitive by default - avw.reporting.defaultIgnoreCaseSearch:** If this checkbox is activated, the checkbox *Ignore Case* on process search mask is activated by default.
- **Exact Id short search only - avw.reporting.exactIdShortSearch:** If this checkbox is activated, you have to enter the right Id to get a correct result.
- **Short search includes subject - avw.reporting.shortSearchSubject:** If this checkbox is activated, the subject will be included in short search.
- **Short search includes process information (forms, history) - avw.reporting.shortSearchFieldvals:** If this checkbox is activated, the following process informations will be included in the short search:
  - process ID,
  - process definition name,
  - task name,
  - start and end date,
  - actor name,
  - organizational unit name,
  - process subject,
  - the same information from each process relation and from each activity instance,
  - process forms and process notes.

  However, it is necessary to initialize a full-text search for every process which should be involved in the search. This can be done with the function in *Admin tasks -> DMS -> Full-text search*. Also, the comments and form field values will be included in short search. This parameter takes effect only if the DMS full-text search is switched on which is only supported when Oracle with LOB or SQL-Server is used as database.
- **Short search includes process documents - avw.reporting.shortSearchDocuments:** If this checkbox is activated, the documents in processes will be considered in a search. This parameter takes affect only if the DMS full-text search is switched on and only if Oracle with LOB or SQL-Server is used as database.

- **Order process Ids by OID - monitoring.orderProcessId:** In worklist and Reporting processes will be sorted by OID, if this checkbox is activated.

  For more information on process relations read the corresponding chapter of the *@enterprise* Application Development Guide.

- **Use underscore (_) as SQL wildcard - avw.reporting.underscoreIsWildcard:** If this checkbox is activated, underscores are allowed as SQL wildcard. If activated, it is not possible to search for '_' unless you escape it yourself.

- **Use smart search algorithm for multi-field searches - list.smartsearch:** If activated, it will be searched globally in all specified fields of a table by using OR-joins. This parameter takes effect on

  – short search in select-list and select-table
  – DOJO object select
  – short search in object-table (e.g. *@enterprise* administration master data tables)
  – short search in form-table
  – short search in select-list of function *Change Agent*

  Example: The search fields are firstname and surname of the user table. In short search field of the user table the string *Roland Eisenberg* has been entered. The sql-condition would be following:

```
(lower(firstname) like(Roland%) or lower(surname) like(Roland%))
 and (lower(firstname) like(Eisenberg%) or lower(surname) like(Eisenberg%))
```

  It is also possible to activate/deactivate this behavior for each table by setting following attribute in configuration file (e.g. myappl.xml):

```
<Attrib key="smartSearch" value="true|false"/>
```

- **Show time in date conditions - avw.reporting.showTimeInDateConditions:** If activated, date and time for datefields on process-/document-searchmask and Reporting condition mask can be entered and on all these masks the appropriate checkbox is activated by default. If this checkbox is deactivated, only the date can be entered as value (without time).

- **Open forms in edit mode - avw.reporting.openFormLinksInEditMode:** Activate this checkbox, if forms in reporting result should be opened in edit mode.

- **Search-delay for ObjectSelects - objectselect.search.delay:** This delay is used in ep/widget/ObjectSelect, if a value is entered. If the delay-period is over, the entered value as yet is sent to the server. Please enter the value in milliseconds (e.g. 0.2s). Duration data type parameter.

3.15 Tuning

With the following parameters the system's performance can be influenced.

- **Worklist Cache - avw.wlcache.state:** Specify, whether the worklist cache should be used. *Activated* means that the cache is used; *Started (but not active)* means that data structures are maintained, but the cache is not used for worklist construction; *Switched off* means that the cache is not used and data structures are not maintained.
- **Do not cache seen objects - avw.wlcache.exemptseenobjects:** If this checkbox is activated, seen objects are not being cached and will be read from the database.
- **Do not cache entry marks - avw.wlcache.exemptmarkedentries:** If this checkbox is activated, entry marks are not being cached and will be read from the database.
- **Do not cache user folders - avw.wlcache.exemptuserfolders:** If this checkbox is activated, user folders of personal worklists are not being cached and will be read from the database.
- **Defer loading of finished parents - avw.wlcache.parents.defer.missing:** If checked, loading of missing parents (for finished parfors, scopes, processes) can be deferred.
- **Restricted Mode when Worklist Cache is not available - avw.wlcache.unavailable.restrict:** If checked, requests to an inactive worklist cache (e.g. during startup) will not fall back to database queries but throw an exception instead. This avoids database overload during cache startup. Recommended for large installations with millions of active activity instances where worklist cache startup may take several minutes. This option will also delay the addition of the HTTP or SSL connector to the embedded Jetty Server until the startup of the worklist cache completes. The Admin connector is always added as soon as possible.
- **Reload classes - avw.class.reload:** Reloads classes without server restart if possible. This should be used only in development environments.
- **Statement statistics - avw.stmt.statistics:** Creates statistics of database statements. If enabled, you will see how often statements have been executed and how much time they consumed (total and average). You can find these statistical information in

  Admin-Tasks → Database connections

  . Don't activate statement statistics for long time periods in production environments because they may need a lot of resources and therefore slow down your server.
- **File cache size (in megabytes) - file.cache.size:** Here you can define the size of the web-server file cache. The default value is `20MB`.
- **Permission Cache activated - aclcache.active:** Check, if the ACLCache should be activated. More details about ACLCache can be found in section ACLCache.
- **Max. number of object specific rights - aclcache.objectrights.maxelems:** Size of the object specific rights cache (in objects).

- **Lifetime of object specific rights - aclcache.objectrights.lifespan.secs:** Lifetime of rights in the object specific rights cache. Duration data type parameter.
- **Max. number of class rights - aclcache.classrights.maxelems:** Size of the class rights cache (in objects).
- **Lifetime of class rights - aclcache.classrights.lifespan.secs:** Lifetime of rights in the class rights cache. Duration data type parameter.
- **ACLCache parameter:** See section ACLCache
- **Inline images into CSS - ep.css.inline.images:** Inlining images (as Base64-strings) causes fewer server requests, but the CSS-file is significantly larger. It depends on your client/network configuration which option has fewer disadvantages.
- **Inline imported CSS-files - ep.css.inline.styles:** Inlining imported CSS-files causes fewer server requests, but the single CSS-file is significantly larger.
- **Allow automatic move into user folders - ep.userfolder.allow.automove:** If this checkbox is activated, worklist entries are moved automatically to appropriate user folders which have entered a XPath expression.
- **No initial listing for the following tables - ep.big.tables:** A comma-separated list of table id's can be entered here where no listing should be performed when displaying table the first time. It is necessary to perform the search function(s) of a table to list the content. Example configuration:
  The string `admin_tree.user,admin_tree.dept` means that no content is listed initially for user table and organizational unit table.

3.15.1 ACLCache

In *@enterprise* it is possible to speed up the rights check by activating the ACLCache. The cache improves the speed of the ACL.hasRight() method calls. The results of calls to method ACL.hasRight() are cached, and the cache is consulted before accessing the database. The cache is organized as an expirable and size bounded LRU cache.

The items have a maximum lifespan associated with them. If an item has been found in the cache, but has expired its lifespan, it is removed from the cache and is reported as being not in the cache. This behavior ensures, that cached right checks do not become unduly outdated. The value lifespan is configurable whereas the default value is 5 minutes.

The cache has also a maximum number of cached elements associated with it. If this number would be exceeded by the insertion of a new cached item, the least recently used item is removed from the cache, thereby ensuring a size bound while providing good hit rate.

Actually, there are two caches, one which stores acl-entries for specific objects and one which stores acl-entries for classes. The parameters for size and lifespan can be configured separately for those two caches.

- **Use partition optimized query for permission checks - acl.separate.targetquery:** ACL evaluations can be tuned by using separate queries for objectscope = 3 versus objectscope <> 3. For this purpose activate this parameter.
- **ACL list: Permission Cache integration - acl.list.cache.usage:** This parameter allows to define how ACL list interacts with ACLCache. Following options are available:
  - `None`: List does not interact with cache
  - `Check only`: Cache is consulted, no results are inserted into cache
  - `Insert positive results only`: Cache is consulted, only positive results are inserted
  - `Full`: Cache is consulted, all results (positive and negative) are inserted into cache)
- **ACL list: Max. number of OIDs in IN-Clause - acl.list.target.splitsize:** The split size for the target set of an ACL.list-query. If the size of the target set is not greater than the split size, *@enterprise* can filter by using a SQL IN-Clause with the target oid's, otherwise a more general filter will be used which may result in a larger result set for that query. In both cases a single SQL statement will be executed. Please note that there are database specific restrictions concerning the number of literals within an IN-Clause and also the textual length of an SQL statement.
- **ACL list: Always restrict by OID for the following classes - acl.list.target.splitclasses:**

  A comma-separated list of fully qualified class names. For those classes, the target set should be splitted so that more than one SQL statement will be executed which always filter by target oid's using an IN-Clause. This is useful if a lot of object specific permissions exists for such a class so that the more general filter would cause a huge result set.

3.16 Security

- **KeyStore file - ssl.keystore:** The Java KeyStore is a binary file, which holds the keys and certificates of the system and the certificates of trusted organizations, so called trust anchors. The KeyStore is the central "database" for certificate management. Ensure that there exists a backup of the KeyStore of *@enterprise*.
- **KeyStore password - ssl.keystore_pwd:** To access a KeyStore a password (with a minimum length of 6 characters) is needed.
- **Default validity period of certificates (days) - cert.default.validity:** If a self-signed certificate should be created, this value is taken for validity period by default.
- **Certificate alias to use for SSL connections - ssl.cert.alias:** Since each user can define his own certificate which is stored in the server KeyStore, the certificate alias to use for ssl connections has to be configured.

- **Password for server certificate - prk.passwd:** The Java API to access the KeyStore is not able to handle different keys with different key passwords. So a system key password has to be configured to access the keys. This password has a minimum length of 6 characters.
- **Bind session to IP address - ep.check.ip:** If activated, the real client ip address is checked with the ip address stored in session.
- **Allow Basic-Auth for WebDAV - avw.dms.allow_basicauth:** Check, if you want to allow Basic-Auth authentication in WebDAV. More information can be found in section Edit Microsoft Office Documents via Browser
- **Open HTML-files in sandbox - ep.dms.html.sandbox:** If this parameter is activated, the execution of scripts and loading of css, images, etc. is prevented when opening HTML-Files in the DMS.
- **Use authentication token in DMS - ep.dms.useauthtoken**: Check, if an authentication token should be passed to Microsoft Office Plug-in. More information can be found in section Edit Microsoft Office Documents via Browser
- **WebDAV authentication class - webdav.auth.class**: Here you may specify an implementation of interface *com.groiss.servlet.WebDAVAuth*. More information can be found in section Edit Microsoft Office Documents via Browser.
- **Allow Basic-Auth for RESTful API - ep.restapi.allow_basicauth:** Check, if you want to allow Basic-Auth authentication for RESTful API.
- **Check Referer header - ep.check.http.referer:** The referer check is enabled by default, means the Dispatcher does not permit requests if the 'Referer' header is missing from the HTTP request, or when it does not match the request. A 'Referer' header matches the request, if the base part of the request URL (consisting of protocol/scheme, host, port and context-root) is a prefix of the 'Referer'.
  Methods and classes marked either as public (via annotation *com.groiss.servlet.Access.mode.Public*), or as entrypoints (via annotation *com.groiss.servlet.EntryPoint*) are never subject to the referer check.
- **Exemptions from Referer check - ep.check.http.referer.exempt:** Additional exemptions from the referer check can be configured here. It is a comma separated list of (fully qualified) Java method names and class names.

3.17 Password policy

The parameters in this section are separable in 3 main groups, which are explained in the following paragraphs.

Note: No parameter of these groups is needed to be set, quite the contrary is recommended. If a too strict password policy is established - especially with the parameters of group 2 -, a brute-force attack may be effective in a small amount of time, because of the insufficient number of possible passwords.
So, if you don't want to set a parameter let the input field blank.

3.17.1 General Policy Settings

The following parameters do not focus on the password itself but on the password change- and login-management. These parameters are:

- **Period of validity (in days) - passwdpolicy.days_password_valid:** Defines the password's period of validity in days.
- **Inform user before password expires (in days) - passwdpolicy.days_warning_before:** Defines the days before the validation time is expired where the user will get a warning at login and - if configured - an email, that his password will expire. An email will be sent only, if a valid mail server is defined in field "SMTP Host" in the section "Communication" of the server configuration (see chapter Communication) and also the timer *PasswordExpiration* has been activated before (see *System Administration Guide* - section Timers for more details).
- **Maximal number of unsuccessful logins until account is deactivated - passwdpolicy.max_count_invalid_logins:** A unsuccessful login is defined as a login attempt of an existing user id with a non valid password. If the specified number of unsuccessful logins are performed between two valid sessions of the specific user, the account is deactivated and the user will get a specific error message on the next login.
- **One-way Hash Algorithm to Use - passwdpolicy.algorithm:** The password is stored in encrypted form by using a one-way-hash function. In former releases this algorithm was the Unix Crypt algorithm. Now one of the following different algorithms can be chosen.
  - **SHA-256 (Secure Hash Algorithm):** Takes a plain string of any length and produces a 256-bit hash output. SHA is said to be secure and is the default value if nothing is configured.
  - **Unix Crypt:** Is limited to 8 bytes input (that means 8 characters), so it is not recommended to use Unix Crypt furthermore. Nevertheless, to ensure compatibility it is supported further on.
  - **SHA-1 (Secure Hash Algorithm):** Takes a plain string of any length and produces a 160-bit hash output. It is not recommended to use this algorithm anymore due to vulnerability!
  - **MD5 (Message Digest 5):** Takes a plain string of any length and produces a 128-bit hash output. MD5 is said to be secure and calculates the hash value faster than SHA.

3.17.2 Default Policy Checker Settings

The release is delivered with a default password checker which ensures proper passwords and which is highly configurable. If you need extended configuration options, it is possible to implement a special password checker.
The following parameters of the default checker can be changed to specify the minimum requirements for a password. The default values are 0!

- **Minimal length of password - passwdpolicy.min_length:** Specifies the minimal length of a password. As an example, if the parameter is set to 8, the password "soccer" is not accepted, but "icehockey" is. (Recommended:4)
- **Maximal length of password - passwdpolicy.max_length:** Specifies the maximal length of a password. As an example, if the parameter is set to 8, the password "hello_its_me" is not accepted, but "hello" is. (Recommended:8)
- **Minimal number of letters in password - passwdpolicy.min_letters:** Specifies the minimal number of letters in the password. As an example, if the parameter is set to 1, the password "1234" is not accepted, but "a1234" is. (Recommended:1)
- **Minimal number of capital letters - passwdpolicy.min_capitals:** Specifies the minimal number of capital letters in the password. As an example, if the parameter is set to 1, the password "hello" is not accepted, but "Hello" is. (Recommended:1)
- **Minimal number of lowercase letters - passwdpolicy.min_lowercase:** Specifies the minimal number of lowercase letters in the password. As an example, if the parameter is set to 1, the password "HELLO" is not accepted, but "hELLO" is. (Recommended:1)
- **Minimal number of digits - passwdpolicy.min_digits:** Specifies the minimal number of digits in the password. As an example, if the parameter is set to 1, the password "Hello" is not accepted, but "Hello1" is. (Recommended:1)
- **Minimal number of special characters - passwdpolicy.min_others:** Specifies the minimal number of special characters in the password. Special characters are defined as any character which does not belong to any of the following character classes: uppercase characters, lowercase characters, digits, space characters. As an example, if the parameter is set to 1, the password "hello" is not accepted, but "hello*" is. (Recommended:0)
- **Minimal number of different characters - passwdpolicy.min_different_chars:** Specifies the minimal number of different characters in the password. As an example, if the parameter is set to 3, the password "aaaa2222" is not accepted, but "aabb2222" is. (Recommended:3)
- **Maximal sequence of same character - passwdpolicy.max_char_adjacent:** Specifies the maximal sequence of the same character in the password. As an example, if the parameter is set to 3, the password "aaaa" is not accepted, but "aaabaaa" is. (Recommended:2)
- **Maximal length of substring - passwdpolicy.max_sub_user_data:** Specifies the maximal length of any substring in the password, which exists in the user's first name, surname or id. As an example, if the parameter is set to 3 and the user's id is "testuser" , the password "stus" is not accepted, but "tes ser" is. This check is case insensitive. (Recommended:2)
- **Number of old passwords to check - passwdpolicy.history_steps:** Specifies the number of old password to check password reuse. As an example, if the parameter is set to 3 and the user changed his password in the order "hello", "itsMe",

"myPassword" and "letMeIn", the password "itsMe" is not accepted, but "hello" is. (Recommended:5)

Note: The history check can only cover old passwords, which have been logged in the database. These old passwords are deleted by the LogTask Timer, so if the timer has deleted all old passwords according to his configuration, the history check can't be performed correctly. The result will indicate a correct password, although the password may have been reused and even be equal to the previous.

- **Minimal number of whitespace characters - passwdpolicy.min_whitespace:** Specifies the number of min. allowed whitespace characters.

Note: If parameters are set in a way that an inconsistent policy is specified, the users may not be able to change their passwords. So please care about the following rules for the parameters:

```
maximal length >= minimal length

minimum capitals + minimum lowercase characters + minimum digits +
minimal special characters <= maximal length

minimum letters + minimum digits +
minimal special characters <= maximal length

minimum different characters <= maximal length
```

3.17.3 Your Own Checker Class

- **Checker Class - passwdpolicy.checker_class:** If the default password checker does not satisfy your requirements, you can enter your own password checker class here. The class must implement the `com.groiss.passwd.Checker` interface.

3.18 Calendar

- **Holiday Class - avw.calendar.class:** Here you can define a class for displaying the holidays in the calendar. It must implement the `com.groiss.cal.Holidays` interface.
- **iMIP - calendar.imip:** If this checkbox is activated, iMIP will be used. In @*enterprise* calendar notifications contain *iCalendar*-files. iMIP offers the possibility to process status information of an appointment.
- **iMIP email address - calendar.imip.email:** Email-address which is used for communicate with the participants; participants will reply to this email-address!
- **Show default resource - cal.show.defaultres:** If this checkbox is checked the user can use a simple resource form for assigning resources to calendar appointments.
- **Resource classes - cal.resources:** It is possible to use arbitrary *Persistent* classes for calendar resources. For this purpose one or more xml nodes can be defined in following way: <xml_id>.<node_id>. The type of the xml node should be a *table* node (see Application Development Guide for more details).

- **Non working day - cal.nonworkingdays:** In this list it is possible to select one or more non working days, which will be needed for example in escalations.
- **Calendar Class - calendar.class:** A calendar class of type `com.ibm.icu.util.Calendar` can be entered here which is used by *@enterprise*.
- **Number of days in agenda-view - calendar.agenda.days:** Define the number of days which are displayed in agenda view (how much days in advance).
- **Start of worktime - cal.worktime.start:** Definition of the time where worktime begins (needed for calculation of process plans).
- **End of worktime - cal.worktime.end:** Definition of the time where worktime ends (needed for calculation of process plans).
- **Worktime per day - cal.worktime.per.day:** Definition of worktime hours (needed for calculation of process plans). This value can be different from time period of start/end worktime. Example: start of worktime is 08:00, end of worktime is 17:00 and worktime per day has value 8.5. The difference between start and end is 9 hours, but the worktime per day is 8.5 hours only (0.5h is the lunch break for example). The calculation in plan tab considers this situation.
- **Calendar sources - cal.applications:** A list of classes can be defined here to activate/deactivate additional calendar-components, e.g. if *com.groiss.calendar.CalendarAppl* is removed, no appointments can be added anymore (default: com.groiss.calendar.CalendarAppl, com.groiss.calendar.wf.DueTasks, com.groiss.calendar.wf.FinishedTasks).
- **Date import formats - cal.impex.formats:** A list of classes which implements the `com.groiss.cal.CalFormat` interface. These classes are used for importing/exporting.
- **Notifier class - cal.notifier:** A class which implements the `com.groiss.cal.Notifier` interface. This class is used for sending notifications (reminder) of due appointments.

3.19 Process cockpit

This section contains the parameters for the process cockpit (see details in the *User Manual*).

- **Root folder - ep.cockpit.rootfolder:** The path to the root folder of the process cockpit.
- **Show overdue processes of last n days - ep.cockpit.deadline.days:** Specifies the number of days, which are used for the calculation of process deadline violations per process definition.
- **Show last n instances - ep.cockpit.recent:** Specifies the number of instances, which are displayed in tab *Runtime* of table *Recently Started* in process cockpit.
- **Common processes - ep.cockpit.commonproc:** A comma separated list of formtypes (Formtype-Id + Version, e.g. jobform_1) which contain the formfield *area*. The forms are used to assign process instances of common processes (for example project) to a cockpit entry.

3.20 Decision Support

This section contains the parameters for the decision support that can be defined for each process definition (see details in the *Administration Guide*).

- **Enable decision support - ml.enabled:** Activates the decision support features. When this is the case, the users can see the *Suggest Values* button as configured and the additional tab in the process definition administration becomes visible.
- **Use cross validation - ml.useCV:** Determines how the learning should be done. When activated, cross validation will be used.

  Otherwise, a simple test/training set split is leveraged.
- **Training set percentage - ml.percentageSplit:** Percentage of data set used for training when using a test/training set split.
- **Number of folds in cross validation - ml.numFolds:** Sets in how many parts the data set should be divided when using cross validation. One part gets used for testing and the remaining ones for training at each iteration.
- **Percentage growing set - ml.rep.percentageGrowingSet:** Which percentage of the test set should be used as the growing set when using reduced error pruning.
- **Penalty for leaf nodes in pessimistic post pruning - ml.pessimisticPostPruner.penalty:** In pessimistic pruning, each leaf node gets penalized with a certain value. The higher the value, the higher the probability gets that nodes get pruned.
- **Example data created since (in days) - ml.showexamples.createdsince.days:** Restricts the sample process instances for a suggested value to those started not before the configured number of days.

3.21 Web components

The @enterprise web application consists of several components. Some of them can be activated or deactivated on demand according to the needs of the particular installation; some of them can be configured to adapt their behavior.
This is achieved via the *@enterprise* configuration file. The deployment descriptor (the *web.xml* file) is a minimal one, the dynamic configuration and component registration is done at application startup by the *EPListener*.
The default *web.xml* file is shipped in the *base/WEB-INF* directory. In a standalone installation with embedded Jetty web server, the *web.xml* file is first searched for in *var/WEB-INF/web.xml*, and if not found there, the default one is used.[2]
The access.log file or the corresponding log files of application servers will contain information about the actions of the listener.
The definition of additional proprietary components (filters/servlets) will still have to be done via editing the web.xml file.

[2] In an installation within an application server, the cointainer will only consult the deployment descriptor at *base/WEB-INF/web.xml*.

- **Context Parameters - ep.webapp.contextparams:** Webapp context parameters in the form of *paramname=paramvalue*.
  The two parameters with the *EPFilter.servlet-names* and *EPFilter.url-patterns* can be used to apply the EPFilter component to your own servlets. The values of the parameters are comma separated lists of servlet names or url patterns.

- **Monitor server with Java Melody - ep.servermonitor.use.melody:** If activated, Java Melody is used as server monitor in *@enterprise*.

- **Monitor DB Connections with Java Melody - ep.dbmonitor.use.melody:** If this parameter is checked, the database connections will be monitored and displayed in *@enterprise* Server monitor (provided that JavaMelody has been activated at all). Please note that in this case, you might have to copy the DB driver jar file to the `base/lib` directory.

- **Init Parameters for JavaMelody - ep.melody.initparams:** It is also possible to define additional parameters for JavaMelody, e.g.:

  - `storage-directory`: Storage directory (path must be absolute) of Java Melody, default is <ep-tmp-director>/javamelody. The default Java Melody directory location is usually specified by the system property 'java.io.tmpdir'. This value can be found e.g. via the corresponding link in the Server-Control of the @ep administration.
  - `warning-threshold-millis` und `severe-threshold-millis`: Thresholds in in ms. These threshold parameters can serve as a basis for a SLA (service level) of an application.
  - `system-actions-enabled`: This parameter (true by default) enables or disables the system actions garbage collector, http sessions, heap dump, memory histogram, process list, jndi tree, opened jdbc connections, database (near the bottom of reports). These actions do require confirmations when necessary.

  These parameters can be used to reduce the memory footprint / general overhead of Java Melody. Issues with monitoring in a production environment could include:

  - Excessive use of file descriptor
  - Memory hogging during startup
  - Some delay during startup

  Sensible initial values for those parameters even in production environment might be:

```
displayed-counters=http,log,error,sql
http-transform-pattern=(?<=/webdav/).*|\\d{10,}
sql-transform-pattern=(?<=[iI][nN] ?)\\([\\d, ]*\\)|\\d{10,}
```

  All these infos are accessible via JMX calls, if the parameter *ep.melody.initparams* is set to value *jmx-expose-enabled=true*. More information of an *@enterprise* server via JMX can be determined via interface `com.groiss.server.ServerInfoMXBean` (see APIDoc for more details). The server must be started with the following parameters:

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.port=<jmx_port> //needed in Tomcat
```

Note: Further information on Java Melody can be found in the *online help*, which can be accessed via a link in the server monitor or at `https://github.com/javamelody/javamelody/wiki`.

- **Use CompressionFilter in Servlets - ep.servlet.use.compression:** If activated, the compression filter is used in @*enterprise*. This filter can, based on HTTP headers in an *HttpServletRequest*, compress data written to the *HttpServletResponse*, or decompress data read from the request. When supported by the client browser, this can potentially greatly reduce the number of bytes written across the network from and to the client.
- **Additional Filter - ep.epfilter.additional:** Provides the ability to define one additional filter dynamically. Here you can enter the class name of it.
- **Init Parameter for Additional Filter - ep.epfilter.additional.initparams:** Parameters for your additional filter.
- **URI Patterns to include in Additional Filter - ep.epfilter.additional.includeuripatterns:** The URI patterns to which your additional filter applies.
- **Exclude "Public" URIs from Additional Filter - ep.epfilter.additional.excludeuripatterns:** If activated, the public URI patterns will be excluded from your additional filter.
- **Enable Process Debugger - ep.process.debugger.enabled:** If activated, the Process Debugger component of @*enterprise* can be used. This component is described in @*enterprise* administration guide, section *Test cases*.
- **Init Parameters for Cometd Servlet - ep.cometd.initparams:** Parameters for the CometdServlet (e.g. transports, timeout , loglevel ,...).
- **Activate Axis Servlet - ep.servlet.use.axis:** If activated, the AXIS2 component in @*enterprise* is used. More details for using web services can be found in @*enterprise* application development guide, chapter *Web services*.
- **Init Parameters for Axis Servlet - ep.axis.initparams:** Parameters for the Axis Servlet.
- **Activate RESTful API Servlet - ep.servlet.use.restapi:** If this parameter is checked the @*enterprise* RESTful API Servlet is activated.
- **Init Parameters for RESTful API Servlet - ep.restapi.initparams:** Parameters for the RESTful API Servlet.

3.22 Other parameters

In this area a various number of helpful parameters are listed. Each parameter has a short description which can be displayed by activating the help icon.

3.23 User authorization via LDAP

This section appears only, if the authorization class `com.groiss.ldap.LDAPPasswdAuth` has been set like described in section Classes. It implements authentication against a directory server. The password check at login is delegated to this server. The *sysadm* user account is exempted from this, it will always be authenticated locally.
There are two general aspects to be configured, namely the technical details of the connection to the LDAP server and the organizational details of the mapping of the @*enterprise* user ids to the LDAP entries. For the communication details, the following items have to be entered:

- **LDAP Host:** The hostname or IP address of the LDAP server.
- **Port:** The port of the LDAP host (default port is 389 for Unencrypted/STARTTLS and 636 for Encrypted).
- **Type of communication:**
  - *Unencrypted:* No encryption used. **Beware**: passwords are transmitted in plain. Recommended only for test environments.
  - *Encrypted:* Standard SSL/TLS encryption. No plain password transfer takes place.
  - *STARTTLS:* Start with plain connection and upgrade it to a secure one. No plain password transfer takes place.
- **Trust level:** Depending on selected communication type one of following trust level must be selected:
  - *System default:* The standard trust mechanisms of Java is being used, this is appropriate when the certificate of the directory server is an official one. Recommended for production use.
  - *Blind:* No real check of server certificate, no check of hostname. While blind trust may be fine for development environments or test purposes, it is strongly discouraged to use it in a production environment.
  - *Certificate in truststore:* The @*enterprise* truststore is being used: if the server certificate has been imported there, it is trusted, even if it is a self signed one.
- **Timeout (ms):** Timeout in milliseconds for communication with the LDAP server. An empty value means no timeout at all.

For the organizational details of the mapping, two general cases can be distinguished:

- **Simple and Flat:** There is a single root entry in your LDAP server to which all the relevant user entries are attached directly. The connection to the LDAP server is initiated with the (presumed) credentials of the user trying to log in. In order to authenticate against such a scheme, just the search path and the user id pattern have to be entered. For such a scheme, leave the *User* and *Password* configuration entries empty and enter the following parameters:

  - Search path: The location in the LDAP tree where the initial connection should be made. Usually the last part of the user pattern, but your mileage may differ, e.g.: ou=Development,dc=acme,dc=org
  - Pattern for User DN: A pattern which leads to the full DN (distinguished name) of the LDAP user entries. The placeholder *${ep_uid}* which is substituted with the id of the *@enterprise* user, e.g.: cn=${ep_uid},ou=Development,dc=acme,dc=org

- **Dispersed or Hierarchical**: There are many different locations where user entries can be found in your LDAP server tree. Such a scheme requires authentication in three stages, namely to first connect with an administrative LDAP user (and her password), to search for an appropriate LDAP entry matching the user id and to rebind the connection with full DN credentials of the found user. For such a scheme, enter all of the following parameters:

  - User: DN of an (LDAP) user allowed to search for the entries, e.g.: cn=Manager,dc=acme,dc=org
  - Password: Password of this user.
  - Search path: The root of the part of the LDAP tree where user entries should be searched, e.g.: ou=Development,dc=acme,dc=org
  - Pattern for User DN: An LDAP filter expression which allows to search for the appropriate user entry. The placeholder *${ep_uid}* is substituted with the id of the *@enterprise* user trying to authenticate, e.g. (&(cn=${ep_uid})(objectClass=inetOrgPerson))

Conditional login according to LDAP group membership can also be accomplished. Let us assume that the organization or the user entries in your LDAP server are dispersed or hierarchical (see above), and that just a subset of all those users entries in the LDAP server are relevant as *@enterprise* account. A common form for such an organization would be to create an entry with object class *groupOfNames* or *groupOfUniqueNames* with e.g. a DN of *cn=epusers,ou=Development,dc=acme,dc=org* that represents the grouping and to add the full DN of the user entries as values of the *member* or *uniqueMember* attributes of the group entry. Then a pattern like
(&(cn=${ep_uid})(objectClass=inetOrgPerson)
(memberOf=cn=epusers,ou=Development,dc=acme,dc=org))
can be used to find the appropriate entry. Please note that the LDAP server has to support this recursive membership searches, a feature that usually has to be configured separately as special module or overlay of the LDAP server.

3.23.1 Transparent Failover with Redundant LDAP Servers

When your infrastructure provides multiple redundant and homogeneously defined LDAP servers, the password authorization can make use of them to ensure transparent failover. Homogeneously defined means that the servers differ only in terms of hostname or ip-address. All other connection parameters and properties must be identical on all of the servers. To configure such a group of servers, enter their hostnames or addresses as a comma-separated list in the **LDAP Host** field, e.g.:

- LDAP Host: ldap1.acme.org,ldap2.acme.org,ldap3.acme.org

All the other properties are to be entered in the same manner as for the single server case as described above.
When the system starts, it marks the first server of the list as the current default one. All password verification attempts will use this server. If the server is not available, the next server in the list is checked (after a timeout, c.f. property **Timeout (ms)**). During one login attempt, there will be at most one connection attempt to each of the LDAP servers. Being *N* the number of entered LDAP hosts, the duration of a login attempt may be *N*Timeout* milliseconds if all your LDAP servers are down.
If the connection to a server has failed, another server is designated as the current default server. During "normal" operation, the last server that has been designated as the default one will be used again and again, until a connection attempt to it fails. This "sticky" behavior ensures that a server recently known to be operational is being tried first with a good probability to succeed thereby without any timeout penalties to pay in the normal case.

3.24 Change administrator password

With this link you can change the password of the *sysadm* user. The corresponding parameter in *ep.conf* is *avw.syspwd*. The default password is *digital* (after a default installation of @*enterprise*).

4 Upgrading your Installation

This chapter describes the upgrading mechanism of *@enterprise*. Some terminology first:

- **Version:** A version is the number of the *@enterprise* version, e.g. *11.0*.
- **Build:** Represents a combination of a version and a revision number, e.g. *11.0.6778*. The revision number can be found e.g. in the *@enterprise* changelog.

To assure the quality and reliability of your installation, bug fixes and enhancements for an *@enterprise* version are provided in regular intervals. New *@enterprise* versions are provided in much larger intervals.
Note that prior to *@enterprise* 11.0 there were different mechanisms depending on whether the version was different between the current system and the new one or if the version was the same but the Build was different - in which case the term 'Upgrade' was used.
Since *@enterprise* 11.0 there is no need for this distinction anymore so the mechanism to upgrade your system is the same in either case.

4.1 Performing an Upgrade of @enterprise

To upgrade your current system first you need to download the newest *@enterprise* installation file `ep-setup-11.0.xxx.jar` in standalone mode (Jetty) or `ep110.war` in an application server (e.g. Tomcat).
Occasionally for special circumstances, we might provide new versions of single artifacts of a revision, but the main distribution format for an upgrade is the installation file.
There are two alternative ways to upgrade your installation. We will first describe the automatic procedure and then a more manual way.

4.1.1 Automatic Upgrade

The automatic upgrade method requires the administrator to place the installation file into a special location (the `upgrades` folder in the `var` area), to stop *@enterprise* to initiate the upgrade by starting *@enterprise* in the upgrade mode, and to restart *@enterprise* normally. What's done automatically is the check of the applicability of the upgrade to the current installation according to the build information as stated above, and to execute the appropriate operations on the file system.

Before an upgrade is performed, a backup of all the files in the `base` area will be made to the backup folder (*upgrades/backup/{timestamp}*) in the `var` area. For each application of an upgrade, a separate backup folder will be created, it will not be deleted by *@enterprise*.
The procedure is as follows:

1. Download the installation file `ep-setup-11.0.xxx.jar` for standalone installation (Jetty) or `ep110.war` for application server (e.g. Tomcat) from our website.
2. It's wise to play it really safe, so we strongly suggest a backup of the installation and the database.
3. Copy the file `ep-setup-11.0.xxx.jar` or `ep110.war` to the `upgrades` folder in the `var` area of the installation. It is required that only installation file is present in that directory.
4. The upgrade procedure can then be initiated as follows:
   - If you are using *@enterprise* in standalone mode (Jetty), you have to start the server in upgrade mode, by stopping *@enterprise* followed by calling the corresponding start script (*epstart.bat* or *epstart.sh* or your own script file) with the single *-upgrade* argument.
     Please note: when running as a Windows service or a Linux daemon, stop the service first.
   - If your installation runs in an Application Server (e.g. Apache Tomcat) use the provided script (epstart.bat/epstart.sh) with option *-upgrade* to apply the upgrade, but in this case the files are replaced only without further automatic action like server restart, DB upgrade or application upgrades.
     Please note: For reasons of file locking, your Application Server, or at least the *@enterprise* application must not be running while performing the upgrade.
     Please note: to use your desired Java version during the upgrade, the path to the directory containing the Java executable can be prepended to the call like stated below. If no such path is provided, some system default Java version will be used.
     - **Windows**: set "JAVAPATH=c:\jdk\x.y\bin" && .\epstart.bat -upgrade
     - **Linux**: JAVAPATH=/usr/opt/java/x.y/bin ./epstart.sh -upgrade
   - If the system is running either as a Windows service or as a Linux daemon, and the particular environment it runs in is rather customized (like e.g. a special account/user the service runs under), then an alternative approach to initiate an upgrade might be better suited.

     Place a file with the name **upgrade.now** into the configured *var* directory for *@enterprise*. The contents of the file do not matter, it can even be empty. When such a file is detected during startup of *@enterprise* the system will perform an upgrade.

     So to initiate the upgrade after creating the 'upgrade.now' file, restart the service or daemon with the usual platform specific actions. The system will perform the upgrade, delete the 'upgrade.now' file and will then attempt to initiate another restart (depending on your service or daemon settings, a manual restart might be needed nevertheless).

Important: if you have customized the locations of `base`, `local` or `var` ensure that these custom settings are also active when performing the upgrade.

5. This action starts the replacement of the files in the base directory of your installation and also applies needed changes to the database.
6. After the upgrade has been successfully applied, the installation file `ep-setup-11.0.xxx.jar` or `ep110.war` is moved to the corresponding backup folder and all actions are logged in the *upgrades/upgrade.log* file in the `var` area.
7. Start *@enterprise* in normal mode by your usual procedure.

Automatic Upgrade for Services

The upgrade procedures described above required to start the system via the extra command parameter *-upgrade*.

This approach is somewhat cumbersome if the system is running either as a Windows service (c.f. section Installing as a Windows Service) or as a Linux daemon (c.f. section Installing as a Linux Daemon).

For such installations, we provide a third possibility to initiate the upgrade. Place a file with the name **upgrade.now** into the configured *var* directory for *@enterprise* and initiate a restart.
When such a file is detected during startup, the system performs an upgrade and then stops the service or daemon.
After that a normal startup of the service or daemon should be performed.
It should go without saying that before using this method, the usual safety procedures such as taking a backup of the installation and the database should be applied.

4.1.2 Manual Upgrade

An alternative approach is to perform all upgrade steps manually. To do so you need to follow the following procedure:

1. Download the installation file `ep-setup-11.0.xxx.jar` or `ep110.war` from our website.
2. It's wise to play it really safe, so we strongly suggest a backup of the installation and the database.
3. Extract the content of `ep-setup-11.0.xxx.jar` into a new directory. We recommend to use the initial setup wizard for this purpose. In case of an application server deploy the `ep110.war` under a new context root.
4. Delete all from the `base` directory in your old installation directory and copy the `base` directory of your new installation directory into the old one.
5. Start the server and login as `sysadm`. You should now be redirected to the upgrade page where you can initiate the necessary upgrade procedure for the database.

4.2 Upgrading an @enterprise Application

The upgrade of an application consists of a set of files which must be replaced in an installation. Typical actions include:

- XML import: Master data of the application can be adapted.
- Execution of database scripts.
- Other JAVA methods.

@enterprise offers two possibilities for upgrading an application, but it depends on the application/the requirements which upgrade procedure should be used:

- **Via defined upgrade path:** *@enterprise* offers a way to define an upgrade path for the application. The path is stored in the file *properties.xml* of the application and should be defined via the *Properties* tab of the application object in the *@enterprise* administration (see *System Administration Guide*, section *Applications*/Tab: Properties). The upgrade path follows the order as defined in the properties file, from top to bottom, with the lowest version listed first and the latest version listed last. We recommend to use this kind to upgrade your application.
- **Via common upgrade method:** *@enterprise* also offers the possibility to execute actions via an upgrade method, if the kind of using the upgrade path is not sufficient. The *upgrade* method is part of the application class which has to implement the interface `com.groiss.wf.ApplicationAdapter`. Further information can be found in the API of *@enterprise* (ApplicationAdapter.getVersion() and ApplicationAdapter.upgrade()).

The upgrade execution itself can be initiated in 2 ways:

- **Via upgrades folder:** Copy the ZIP file created with the administration function *Export application* into the *upgrades*-folder in the *var* area of your *@enterprise* server (see manual *System administration guide*, section *Export application*). Start the server with the *-upgrade* option. Instead of performing these steps manually we recommend to use the administration function *Install/Upgrade application* under *Admin tasks - Import/Export* - see System Administration Guide for more details!
- **Via Upgrade button on tab General of an application object:** Extract the application JAR file from the ZIP archive and copy it to the *lib* folder on application file system. Start server without *-upgrade* option. Perform login as sysadm, change to the application object, open the tab *General* and perform the upgrade by executing button *Upgrade* (see System Administration Guide, section *Applications*).

4.3 Performing an Upgrade of @enterprise prior to 11.0

This section describes the steps needed to upgrade an *@enterprise* installation older than version 11.0 to version 11.0. Therefore it is required to understand the new directory structure of an *@enterprise* installation since version 11.0 which is described in section File structure

of stand-alone server respectively section File structure in an Application Server or Servlet Container. With this in mind it should be clear to find the correct location for your custom files and directories which you want to reuse in the new version.

The procedure is as follows:

1. Backup your old installation and database.
2. Extract the content of `ep-setup-11.0.xxx.jar` into a new directory. We recommend to use the initial setup wizard for this purpose.
3. If you want to outsource the `local` and `var` area from the installation directory create the root directories for these areas and ensure to set the corresponding environment variables `EP_LOCAL` and `EP_VAR` accordingly, e.g. in your own start script.
4. Copy your existing custom files and directories into the corresponding areas of the new version, e.g.
   - your *@enterprise* server configuration file(s) to `var/conf`[1],
   - your applications to `local/appls`[2],
   - your `forms` directory to `var/forms`,
   - your custom additional jar files currently located in the `lib` or your server installation to `local/lib` (e.g. your JDBC driver jar file)

   For any file and directory which default location has been changed via configuration this is an optional step, e.g. your `forms` directory. If you want to keep those files and directories at their current location you can do so, but we recommend to migrate to the new file structure.
5. If you are using an embedded database (H2 or Derby) located within your old installation directory it is recommended to copy the database directory into the var area of the new installation and change the path to the database in your configuration file which is stored as part of the value of property `database.url`:
   - H2: `jdbc:h2:./h2db/ep;...` -> `jdbc:h2:${EP_VAR}/h2db/ep;...`
   - Derby: `jdbc:derby:ep;...` -> `jdbc:derby:${EP_VAR}/ep;...`
6. To perform an interactive upgrade, start the server and login as `sysadm`. You should now be redirected to the upgrade page where you can initiate the necessary upgrade procedure for the database.

[1] If your configuration file has the name `avw.conf` you need to rename it to `ep.conf`. It you had several configuration files, you need to set the `EP_CONFFILES` environment variable.

[2] If the application directory of an application is not located within the `appls` directory of your old installation the specification of that directory need to be changed also in your database

Hint: The following exception will be prompted when starting the server - either for manual or unattended upgrade:

```
com.groiss.component.Configuration not a number: ep.forms.generate.jar=true
   java.lang.NumberFormatException: For input string: "true"
   ...
```

This has no consequences for the upgrade, in fact the reported property will be migrated during the upgrade so that consecutive server starts will not prompt that exception again.

7. To perform an unattended upgrade, start the server with the *-upgrade* option. Any required database upgrades will be performed. The server will be stopped automatically.[3]

8. (Re-)Start the server and delete Browser caches.

Hint: Whether or not you have moved your applications into `local/appls`, after the successful upgrade for each application its configuration file `appl.prop` has been copied to `var/appls/<application-id>` and will now be loaded from that location. So when changing some configuration properties of an application the changed values will be stored in this file at its new location. But the original `appl.prop` file is still needed at its old location too as information like the application's Id may be read from that file when upgrading the application at a later point in time.

4.3.1 Migrating custom start scripts

Prior to *@enterprise* 11.0 is was quite common to modify the shipped start script (`ep.bat`, `ep.sh`) or at least to copy parts of it into a custom script. This should not be necessary anymore when using the new environment variables and argument files described in File structure of stand-alone server. So check your modifications and move your custom java properties to file `localjavaargs` or `varjavaargs` and your configuration file(s) parameter to environment variable `EP_CONFFILES` and just call the shipped start script as shown in the example in Customization.

4.4 Migration of deprecated DBMS features

4.4.1 Migration of Oracle data types LONG and LONG RAW

This section describes the steps to migrate the Oracle data types LONG and LONG RAW, which have been deprecated since Oracle version 9i.
If existing installations continue to use the old data types, certain operations might not work or certain features might not be available.

[3] If you are using PostgreSQL as your DBMS, it might be necessary to use this upgrade variant and refrain from using the interactive one.

Hint: Please note that the migration is at your own risk, can take a long time and require additional storage space! The steps below act as a only guideline. Do play it save and have appropriate, current and working backups and stage the migration to gain experience. Nevertheless, we strongly recommend to migrate to the new data types as follows:

1. Backup your old database!

2. Write down the indexes of the affected tables. Following query helps to get a list of the affected indexes wrapped in executable *alter index* statements:

```
select 'alter index '||index_name||' rebuild;' as command
from user_indexes where table_name in (
  select table_name
  from user_tab_columns
  where (table_name like 'AVW%' or table_name like 'FORM%')
  and data_type  in ('LONG', 'LONG RAW')
  and index_type <> 'DOMAIN'
)
order by index_name;
```

   Do not execute the *alter index* statements generated in this step! Save the statements for later execution.

3. If your installation uses full text search and the indexed tables use the old data types, then save your current full text index definitions and then drop those indexes. A list of such indexes can be obtained by:

```
select index_name
from user_indexes
where index_type = 'DOMAIN'
order by index_name;
```

   Since your full text index definitions might be arbitrarily complex, we cannot provide a universally applicable statement here. For the usual full text indexes in @enterprise, the following statements are sufficient:

```
drop index avw_ctxdoccont;
drop index avw_ctxfieldvals;
drop index avw_ctxprocfieldvals;
```

4. For each table which contains one of the previous mentioned data types an *alter table* statement must be performed like in following way:

   - alter table <tn> modify (<longcolname> clob default empty_clob());
   - alter table <tn> modify (<longrawcolname> blob default empty_blob());

   It is a little bit cumbersome to get the affected tables. For this purpose, the following query helps to get a list of affected tables wrapped in executable *alter table* statements as mentioned above:

```
select
   'alter table '||table_name||
   ' modify ('||column_name||' '||
    (case when data_type='LONG' then 'CLOB' when data_type = 'LONG RAW'
     then 'BLOB' else 'ERROR' end)|| ' default empty_'||
    (case when data_type='LONG' then 'CLOB' when data_type = 'LONG RAW'
     then 'BLOB' else 'ERROR' end)||'());'
    as command
    from user_tab_columns
where (table_name like 'AVW%' or table_name like 'FORM%')
and data_type  in ('LONG', 'LONG RAW')
order by table_name, column_name;
```

Now execute those statements.

5. Perform the *alter index* statements which have been created in step 2 above to rebuild the indices.

6. If your installation uses full text search, then recreate them. Again, since your full text index definitions might be arbitrarily complex, we cannot provide universally applicable statements here. For the usual full text indexes in @enterprise, the following statements are sufficient:

```
execute ctx_ddl.create_preference('case_insensitive', 'BASIC_LEXER');
execute ctx_ddl.set_attribute('case_insensitive', 'mixed_case', 'NO');

create index avw_ctxdoccont on avw_doccontent(content)
indextype is ctxsys.context parameters('lexer case_insensitive');

create index avw_ctxfieldvals on avw_formfieldvals(fieldvalues)
indextype is ctxsys.context parameters('lexer case_insensitive');

create index avw_ctxprocfieldvals on avw_procfieldvals(fieldvalues)
indextype is ctxsys.context parameters('lexer case_insensitive');
```

7. Now, you can check your installation by executing the following two selects which should have empty results:

```
select index_name, status
from user_indexes
where index_type <> 'DOMAIN'
and status<> 'VALID';

select index_name, status, domidx_status, domidx_opstatus
from user_indexes
where index_type = 'DOMAIN'
and (domidx_status <> 'VALID' or domidx_opstatus <> 'VALID');
```

and also the select statement from step 4 above should now return no rows.

4.4.2 Migration of Oracle Storage Type for LOBs

Beginning with version 11g, Oracle introduced a new mechanism for storing large objects (LOBs) in the database. Previously "BASICFILE" had been used, the new approach is called "SECUREFILE". Oracle encourages its customers to migrate to SECUREFILES.
The features, benefits and possible peculiarities of SECUREFILE have been dealt with elsewhere (e.g. https://www.oracle.com/technetwork/database/features/secure-files/securefiles-whitepaper-2009-160970.pdf).
We found SECUREFILES to avoid certain annoying situations with BASICFILES where a simple update of a single (and quite small) BLOB may appear to hang for a long time (minutes). This behavior is indeterministic, not reproducible in general but may give the impression that the application is hanging while waiting for the completion of the update in Oracle.
If you experience strange long waiting times (especially when inserting into `avw_log` or updating `avw_doccontent`), you can try to determine if there were lots of waits around the problematic points in time:

```
select session_id, sample_time, session_state, event, wait_time, time_waited,
   sql_id, sql_child_number CH#,
   current_obj#, current_file#, current_block#
from v$active_session_history
where
user_id = (select id from all_users where user_name='EPUSER') and
sample_time between
    to_date('2019-08-02 14:18:00','yyyy-mm-dd hh24:mi:ss')
       and
    to_date('2019-08-02 14:30:00','yyyy-mm-dd hh24:mi:ss')
   and   time_waited > 0
   AND current_obj# <> -1
order by sample_time, session_id;
```

The sql text associated with the sql_ids can be obtained via:

```
 select sql_text from v$sqlarea where sql_id='<sql_id>';
```

and together with the event type and schema objects involved gives a hint about the nature of the problem.
In *@enterprise* the following configuration settings can also help to pinpoint the problem:

- Configuration/Database/DB connection reservation warning interval :
  threshold for transaction duration. Automatic logging of stack traces of long running operations will take place. Duration data type parameter.
- Configuration/Logging/Custom loglevels:
  com.groiss.dms.store.DocumentContent=TRACE

If your installation experiences similar symptoms, check if you are using BASICFILES for storage and consider migrating to SECUREFILES.

@enterprise does not consider any system specific physical data aspects. When creating tables with LOBs, the default storage system (cf. "db_securefile") in your instance will be used.

```
show parameter db_securefile
```

A mixture of storage types could exist, e.g. if older tables had been created with BASICFILES and newer ones with SECUREFILES. To determine which of your tables has not yet SECUREFILES storage for lobs:

```
select table_name
from dba_lobs
where owner = 'EPUSER'
and (table_name like 'AVW%' or table_name like 'FORM%')
and securefile='NO'
order by table_name;
```

Migration Considerations

- **General:** The migration is a process completely internal to your Oracle installation, no APIs or other operational procedures need to be changed.

  We will restrict ourselves sketching the migration using the online redefinition package.

  Please note that the steps below do just act as a guideline. Do play it save and have appropriate, current and working backups, stage the migration to gain experience and also follow general advice from Oracle.

- **Availability:** During online redefinition, your DB will be available in principle, but the migration is very io-intensive, so its advisable to do it in non-prime time.

- **Storage space:** Migration does need a lot of space. The space for the original data as well as a comparable amount of space for the migrated data will be used. Redo space is also needed. If this is a problem for your installation, please consider other approaches (like exporting and then importing into a table with an altered definition).

  ```
  select sum(length(MYLOBCOLUMN)) from MYTABLE;
  ```

- **Time:** Migration can take a considerate amount of time. We experienced run times of about 2 minutes per GB of lobs on quite moderate hardware with almost no other activity in the system.

Generating SQL statements for online redefinition:

```
select 'exec DBMS_REDEFINITION.redef_table(uname => '''||owner||
   ''', tname => '''||table_name||
   ''', lob_store_as =>''SECUREFILE'');'
from dba_lobs
where owner = 'EPUSER'
and (table_name like 'AVW%' or table_name like 'FORM%')
and securefile='NO'
order by table_name;
```

The generated statements have to be executed![4] Afterwards please check if any indexes got unusable during the operation:

[4] You might get a DRG-11439 error from Oracle for tables that have a full text ("domain") index. In this case, drop and recreate the domain indexes after performing the redefinition.

```
SELECT owner, index_name, tablespace_name
FROM   dba_indexes
WHERE  status = 'UNUSABLE';
```

and fix them via the following generated statements as needed:

```
SELECT 'alter index '||owner||'.'||index_name||' rebuild tablespace '||tablespace_name ||';'
FROM   dba_indexes
WHERE  status = 'UNUSABLE';
```

4.4.3 Migration of deprecated MS SQL-Server data types

Since MS SQL-Server 2005 Microsoft has been set some data types to deprecated and replaced them by new ones. The following list contains the deprecated and the appropriate new data types:

- text has been replaced by varchar(max)
- ntext has been replaced by nvarchar(max)
- image has been replaced by varbinary(max)

Existing installations can continue to use the old data types, but new installations should use the new data types. This section describes the recommended migration steps for existing installations with MS SQL-Server 2005 and newer:

Hint: Please note that the migration is at your own risk and can take a long time!

1. Backup your old database!
2. For each table which contains one of the previous mentioned data types an *alter table* statement must be performed like in following way:
   - alter table <tn> alter column <textcolname> varchar(max);
   - alter table <tn> alter column <ntextcolname> nvarchar(max);
   - alter table <tn> alter column <imagecolname> varbinary(max);

   It is a little bit cumbersome to get the affected tables. For this purpose, the following query helps to get a list of affected tables wrapped in executable *alter table* statements as mentioned above:

```
select
   'alter table '+table_name+
   ' alter column '+column_name+' '+
   (case when data_type='text' then 'varchar(max)'
         when data_type='ntext' then 'nvarchar(max)'
         when data_type = 'image' then 'varbinary(max)' else 'ERROR' end)+
   ';'
   as command
   from information_schema.columns
where (table_name like 'AVW%' or table_name like 'FORM%')
and data_type  in ('text','ntext', 'image')
order by table_name, column_name;
```

Heterogeneous data types for %OIDTYPE% patterns

Installations using MS SQL-Server as database engine might suffer from heterogeneous types being used for %OIDTYPE% patterns.

The old DB translator (used for SQLServer version < 2000) uses "DECIMAL(20)", the new one (for SQLServer versions >= 2005) uses "BIGINT". In really ancient installations, the type "DECIMAL(28)" might also be in use.

As a consequence, there might be installations with more than one type being used for oid columns or columns referencing them. This has not been a problem up to recently.

But in *@enterprise* 9.0 we began to use referential integrity constraints (foreign keys) in our schema and were hurt by a peculiar behavior of SQL-Server which insists that the data types of referencing columns and referenced columns must be exactly the same.

So, depending on the configuration and history of the installations there might be problems when upgrading to a recent *@enterprise* version because of nonuniform data type usage.

To remedy this potential problem, two measures have been implemented:

1. The new translator has been extended to determine the oidtype at startup. Essentially, if the oid columns of all avw* and form* tables are using the same data type, then this data type is being used as %OIDTYPE%. If not, BIGINT will be used.

   With this change, all installations which have not yet switched to the new translator might do so without being affected by the issue.

2. For installations which already do use different data types for %OIDTYPE%, we provide a powershell script which generates the needed SQL DDL for type migration to BIGINT.

   Please contact support@groiss.com for further information regarding the script.

5 Containerized @enterprise (Docker)

For deployment in a containerized environment, *@enterprise* supports the following specific functionality

- waiting at startup until the configured database service is available,
- initialization of the *@enterprise* database schema (if necessary),
- updating the *@enterprise* database schema (if necessary),
- adding applications at build-time and installing them automatically at runtime,
- upgrading of files of already installed applications at build-time and upgrading application data at runtime

We do not deliver ready to use docker images, but choose to provide you with a cookbook how to create them on your own. So you can use base images of your choice, i.e. e.g. which Linux distribution and which Java version and distribution should be used.

Hint: As base image for the following example Dockerfiles we use `eclipse-temurin:17-jre-alpine` from docker hub. This is an official image maintained by Adoptium that is a viable base.[1]

5.1 Creation of images and run containers

As preliminary steps before delving into the examples, please

- create a directory `dockerroot` on your host,
- download the `ep-setup-11.0.xxx.jar` file for *@enterprise* from our download portal and save it in `dockerroot`,
- from `ep-setup-11.0.xxx.jar`, extract the content of folder `docker` in file `demos.zip` (which is located in directory `base/doc/examples`), to `dockerroot`

All files referenced in the following sections can be found in directory `docker` of file `demos.zip`.

[1] Please note that naturally we cannot assume a guarantee for that image - it is your responsibility to decide if you want to use this image or choose a different one.

5.1.1 Minimal example

First let's start with a minimum viable example of a Dockerfile which creates an image containing *@enterprise* and starts the *@enterprise* server when the container is started.

File **docker/Dockerfile-ep**

```
# docker file installing ep in alpine linux with a jre 17
FROM eclipse-temurin:17-jre-alpine
LABEL maintainer "support@acme.com"

# Setup EP_HOME -- useful when modifying the image in another Dockerfile
ENV EP_HOME /home/ep

# set the working directory for all commands
WORKDIR ${EP_HOME}

# mount the host directory containing the ep-setup file and start the installation
RUN --mount=target=/tmp/ep-files java -jar /tmp/ep-files/ep-setup-*.jar . default 0

# make the default port accessible
EXPOSE 8000

# start @enterprise
CMD ["sh", "base/epstart.sh"]
```

To create the image, please execute the following command from within the `dockerroot` directory:

```
docker image build -f Dockerfile-ep -t acme/epdemo .
```

Docker will pull the base image, enrich it by extracting the `ep-setup-11.0.xxx.jar` file and store it under the name `acme/epdemo`. You can check the image with one of the following commands:

```
docker image list --all
docker image inspect acme/epdemo
```

A container from that image can be started via

```
docker container run --name epdemo -p 8000:8000 -d acme/epdemo
```

When navigating to `http://localhost:8000` with a web browser, you should see the first page of the *@enterprise* setup wizard. After finishing the *@enterprise* setup wizard your *@enterprise* server is ready to use [2].
A shell session to the container can be established via execution of

```
docker container exec -it epdemo /bin/sh
```

The shell prompt will indicate that you are in the `/home/ep` directory of the container. You can e.g. list the contents of the access log via `less var/log/access*.log` [3]. The shell can be terminated by `exit`.
You can stop the running container using the following command from the host

[2] For the steps of the setup wizard, please see section Extract and Install
[3] Type key `q` to exit `less`

```
docker container stop epdemo
```

and start it again at any time by calling

```
docker container start epdemo
```

Running containers can be listed via

```
docker container list
```

All containers can be listed via

```
docker container list --all
```

If a (stopped) container is not needed anymore, it can be removed by

```
docker container rm epdemo
```

But beware: if you remove the container, all changes done by the setup process are gone! You will need to execute the setup wizard again when starting a new container, as all changes were only stored in the writable layer of your old container. So the scenario above is only useful to test if your image works.

Hint: If the browser could not establish a connection, most likely your container could not be started successfully. As a first step execute command

```
docker container ps
```

to check if the container is running. If your container is not listed by this command, you can try to start the container in interactive mode via one of the following command:

```
docker container run --name epdemo -it acme/epdemo /bin/sh
```

You should now see a shell prompt in which you can try to start the @*enterprise* server by calling

```
sh base/epstart.sh
```

Any error occurring during startup should now be prompted to the console, which should help you to identify and solve the problem.

5.1.2 Example with a preconfigured server

More relevant than the previous example is the case in which you provide an already populated @*enterprise* configuration file. Upon the first start of the container, the database is created and set up, so no additional manual set up steps using the wizard will be needed.
Your `dockerroot` directory contains an @*enterprise* configuration file `ep.conf` in which the basic properties are set, like:

- license key
- password for the sysadm user

- localization settings
- database settings
- ...

Please provide valid values for the placeholders of parameters `avw.license` and `avw.syspwd`. Additionally you need to download the H2 driver jar - either from `http://www.h2database.com` or a maven repository - and copy that jar file also into directory `dockerroot`.

Then stop and remove the old container named `epdemo` as we want to reuse the name for our new container and execute the following command (as one line):

```
docker container run --name epdemo -p 8000:8000 -d \
   --mount type=bind,source="$(pwd)"/h2-2.2.224.jar,target=/home/ep/local/lib/h2-2.2.224.jar \
   --mount type=bind,source="$(pwd)"/ep.conf,target=/home/ep/var/conf/ep.conf \
   --mount type=bind,source="$(pwd)"/h2dbfiles,target=/home/ep/var/h2db \
   --mount type=bind,source="$(pwd)"/logfiles,target=/home/ep/var/log \
   acme/epdemo
```

The source on the host machine must be specified as absolute path in the `mount` options, so if your host runs with Linux you can use `$(pwd)` to reference the current working directory. In case of Windows you can use `%cd%`:

```
docker container run --name epdemo -p 8000:8000 -d ^
--mount type=bind,source="%cd%"/h2-2.2.224.jar,target=/home/ep/local/lib/h2-2.2.224.jar ^
   --mount type=bind,source="%cd%"/ep.conf,target=/home/ep/var/conf/ep.conf ^
   --mount type=bind,source="%cd%"/h2dbfiles,target=/home/ep/var/h2db ^
   --mount type=bind,source="%cd%"/logfiles,target=/home/ep/var/log ^
   acme/epdemo
```

Hint: If you want to use an already existing configuration file of yours you need to add the following line to that configuration file to enable the automatic database schema initialization:

```
ep.autodeployment.enabled=true
```

If you now open a browser and navigate to localhost:8000 you will see the *@enterprise* login screen. Log on as sysadm and you will see that the *@enterprise* database schema has been loaded and your server is now ready to be used.

When checking your current directory, you will see that it now contains two new sub-directories:

- h2dbfiles: contains the files created by the H2 database to store its data
- logfiles: contains the *@enterprise* access and error log files written by the *@enterprise* server run in your container

By mounting those directories into your container, the files created/changed in those directories by the running *@enterprise* server will be written on file system of the host and not in the writable layer of the container. Therefore these data will remain even when you remove your current container. The same is valid for the mounted configuration file. If

you perform any change in the *@enterprise* server configuration using the Administration UI of *@enterprise* that changes will be written to the configuration file `ep.conf` on your host.

To demonstrate this, create a new user via the Administration UI of *@enterprise*. Then execute the following commands:

```
docker container stop epdemo

docker container rm epdemo

docker container run ... ( like above)
```

Via the Administration UI you can see that the created user still exists as the database files are taken from directory `h2dbfiles` of your host system.

Hint: In the example above, we have also mounted the database driver file (for H2 in this case) into the container as this file is not part of the image we created before. Alternatively you can copy that jar file into the image by modifying the Dockerfile or by creating your own image based on the original image - which is most likely the preferred way, especially for productive images. We will show an example for this later on.

5.1.3 Example with docker-compose

It is annoying and error prone to type all those `mount` options again and again - and most likely you won't remember them when time goes by. If not managing your containers using tools like Kubernetes, OpenShift or others, an alternative is to specify all those options via a docker-compose file. `demos.zip` contains file `docker-compose-ep.yml` in its directory `docker` as an example for such a file:

File **docker/docker-compose-ep.yml**

```
# Start ep container via docker-compose
services:
  epdemo:
    image: acme/epdemo
    container_name: epdemo
    ports:
      - "8000:8000"
    volumes:
      - type: bind
        source: ./h2-2.2.224.jar
        target: /home/ep/local/lib/h2-2.2.224.jar
      - type: bind
        source: ./ep.conf
        target: /home/ep/var/conf/ep.conf
      - type: bind
        source: ./h2dbfiles
        target: /home/ep/var/h2db
      - type: bind
        source: ./logfiles
        target: /home/ep/var/log
```

Creating and starting a container using docker compose is done by the following command (if you still have the previous container, first remove it via `docker container rm epdemo`):

```
docker-compose -f docker-compose-ep.yml up -d
```

You can also stop the container and start it again later on with the following commands:

```
docker-compose -f docker-compose-ep.yml stop

docker-compose -f docker-compose-ep.yml start
```

Or you can stop and remove the container with this command:

```
docker-compose -f docker-compose-ep.yml down
```

Nevertheless we use the `docker` command instead of `docker-compose` for our following examples.

5.2 Inclusion of applications in the image

So far we saw how to create an image for a vanilla @*enterprise* server and how to run a container of that image. Most of the time you will also want to run @*enterprise* applications in your @*enterprise* server. In this section we will show how to create images containing such applications.

5.2.1 Inclusion of one application

We will use the image created in the section before and enrich it with the @*enterprise* application 'Human Resources Processes' which is freely available as `hr.zip` for @*enterprise* customers in our download portal. After downloading the application you need to copy that zip file into the `dockerroot` directory. Another change to our base image is that this time we incorporate the database driver file into the image instead of mounting it.

File **docker/Dockerfile-ep-hrp**

```
# docker file based on image created with Dockerfile-ep
# and installing application 'Human Resources Processes'
FROM acme/epdemo
LABEL maintainer "support@acme.com"

# copy the database driver into the local lib directory of ep server
COPY ./h2-2.2.224 ${EP_HOME}/local/lib/h2-2.2.224.jar

# extract the application into appls-dir using a mount to the build context
RUN --mount=target=/tmp/ep-files unzip /tmp/ep-files/hr.zip -d local/appls/hr
```

Again we use docker's `build` command to build our new image:

```
docker image build -f Dockerfile-ep-hrp -t acme/epdemo-hrp .
```

The next step is to create an empty file named `appl.prop` in directory `dockerroot`. Now start a container of this new image using the following docker run command on Linux:

```
docker container run --name epdemo-hrp -p 8010:8000 -d \
   --mount type=bind,source="$(pwd)"/ep.conf,target=/home/ep/var/conf/ep.conf \
   --mount type=bind,source="$(pwd)"/h2dbfiles,target=/home/ep/var/h2db \
   --mount type=bind,source="$(pwd)"/logfiles,target=/home/ep/var/log \
   --mount type=bind,source="$(pwd)"/appl.prop,target=/home/ep/var/appls/hr/appl.prop \
   acme/epdemo-hrp
```

or this one on Windows

```
docker container run --name epdemo-hrp -p 8010:8000 -d ^
   --mount type=bind,source="%cd%"/ep.conf,target=/home/ep/var/conf/ep.conf ^
   --mount type=bind,source="%cd%"/h2dbfiles,target=/home/ep/var/h2db ^
   --mount type=bind,source="%cd%"/logfiles,target=/home/ep/var/log ^
   --mount type=bind,source="%cd%"/appl.prop,target=/home/ep/var/appls/hr/appl.prop ^
   acme/epdemo-hrp
```

Note the changes in these commands with respect to the one used to start a container of our vanilla *@enterprise* image:

- port 8000 in the container is mapped to port 8010 on the host to avoid conflicts with our previous examples
- the database driver file is not mounted anymore as it is now part of the image
- a mount for the application's configuration file `appl.prop` is added to preserve changes made in the application's configuration section

When navigating to `http://localhost:8010` and log on as sysadm you will see that the application is now available in your *@enterprise* server.

5.2.2 Installation order

When multiple applications are installed, it may be necessary to define the sequence in which the applications shall be set up and also be started. By default the sequence is determined by alphanumerically sorting the Ids of the applications. Alternatively you can define the sequence via property `ep.autodeployment.applications.sequence` in the *@enterprise* configuration file `ep.conf`, e.g.

```
ep.autodeployment.applications.sequence=hr,another_app
```

In this case application `hr` will be set up first, followed by application `another_app`. This order will also be taken for all consecutive starts of the *@enterprise* server as well as for the upgrade of the applications.

5.2.3 Additional install instructions

There may be cases in which you need to perform additional actions which cannot be handled by the XML imports registered in the `appl.prop` of your application - e.g. when you want to import test users in your test environment. To support this, your image may contain an installation script in the form of a file called `install.scr` at location `$EP_HOME/local/appls`. This file is a groovy script in which you can use the *@enterprise* API to perform e.g. the import of such users:

File **docker/install.scr**

```
//import test departments
admin.importXML("demodepts.xml");
//import test users
admin.importXML("demousers.xml");
```

The mentioned XML files are in the `hr.zip` file; the `install.src` file is from directory `docker` in `demos.zip` Copy those files directly into the `dockerroot` directory.
There are two possibilities how to make the installation script and the XML files accessible to the container. You can change the Dockerfile for image creation and add the following line at the very end:

```
COPY ./demodepts.xml local/appls/hr/classes/demodepts.xml
COPY ./demousers.xml local/appls/hr/classes/demousers.xml
COPY ./install.scr local/appls/install.scr
```

Or you can mount the required files when running a container so that you could use your image for production as well as for test environments without polluting your production image with test related data:
That is, in Linux use:

```
docker run --name epdemo-hrp -p 8010:8000 -d \
 --mount type=bind,source="$(pwd)"/ep.conf,target=/home/ep/var/conf/ep.conf \
 --mount type=bind,source="$(pwd)"/h2dbfiles,target=/home/ep/var/h2db \
 --mount type=bind,source="$(pwd)"/logfiles,target=/home/ep/var/log \
 --mount type=bind,source="$(pwd)"/appl.prop,target=/home/ep/var/appls/hr/appl.prop \
 --mount type=bind,source="$(pwd)"/demodepts.xml,target=/home/ep/local/appls/hr/classes/
  demodepts.xml \
 --mount type=bind,source="$(pwd)"/demousers.xml,target=/home/ep/local/appls/hr/classes/
  demousers.xml \
 --mount type=bind,source="$(pwd)"/install.scr,target=/home/ep/local/appls/install.scr \
 acme/epdemo-hrp
```

and under Windows:

```
docker run --name epdemo-hrp -p 8010:8000 -d ^
 --mount type=bind,source="%cd%"/ep.conf,target=/home/ep/var/conf/ep.conf ^
 --mount type=bind,source="%cd%"/h2dbfiles,target=/home/ep/var/h2db ^
 --mount type=bind,source="%cd%"/logfiles,target=/home/ep/var/log ^
 --mount type=bind,source="%cd%"/appl.prop,target=/home/ep/var/appls/hr/appl.prop ^
 --mount type=bind,source="%cd%"/demodepts.xml,target=/home/ep/local/appls/hr/classes/
```

```
 demodepts.xml ^
--mount type=bind,source="%cd%"/demousers.xml,target=/home/ep/local/appls/hr/classes/
 demousers.xml ^
--mount type=bind,source="%cd%"/install.scr,target=/home/ep/local/appls/install.scr ^
acme/epdemo-hrp
```

But note: this script will only be called if at least one application is installed in directory `appls` and no application is already available in the database used for your *@enterprise* server.

5.2.4 Upgrading *@enterprise* and/or applications

Whenever a new revision of *@enterprise* or any of the applications shall be rolled out, this simply means to replace the `ep-setup-11.0.xxx.jar` file and/or application zip files in your docker build environment and build a new image using the existing Dockerfiles. Then you stop your running container, remove it and run a new one using the new image.

Any needed updates to the *@enterprise* database schema will be executed automatically. The same applies for any application already existing in the old image.

New applications will be detected and installed automatically, but as already mentioned: the script `install.src` will NOT be executed again. Removed applications will **not** be deleted from the database schema.

Note that these automated steps will only be performed when configuration parameter `ep.autodeployment.enabled` is set to `true`.

5.3 Preserving runtime data

As already mentioned, all data that is created and/or changed by the *@enterprise* server running within a container is lost after removing that container if not mapped to an external volume - e.g. by using the `-mount` option as shown in the examples above. For a productive server all relevant runtime data is stored in the `var` directory of the *@enterprise* server - besides the data that is stored in the database. So most likely instead of defining multiple mounts to the files and directories in that area a single mount of the whole `var` directory would be the easiest way to preserve that data.

Note that in this case you need to copy file `ep.conf` located in `dockerroot` to directory `conf` in the mounted `var` directory, otherwise the *@enterprise* server will show the setup screen in the browser as no configuration can be found.

Linux:

```
docker container run --name epdemo-hrp -p 8010:8000 -d \
   --mount type=bind,source="$(pwd)"/var,target=/home/ep/var \
   acme/epdemo-hrp
```

Windows:

```
docker container run --name epdemo-hrp -p 8010:8000 -d ^
   --mount type=bind,source="%cd%"/var,target=/home/ep/var ^
   acme/epdemo-hrp
```

The only questionable directory is `var/tmp` which may not be considered as a candidate for data that needs to survive the lifetime of its container. If you want to exclude that directory, you would need to mount all other directories in `var` by a mount option and just spare the `var/tmp` directory.

A better alternative would be to mount directory `var` as a whole, but define another mount for the `var/tmp` directory. In that case the `var/tmp` directory can be mapped to a directory which will not be backed up and maybe will be cleaned up automatically every time the container will be replaced by a new one.

Linux:

```
docker container run --name epdemo-hrp -p 8010:8000 -d \
   --mount type=bind,source="$(pwd)"/var,target=/home/ep/var  \
   --mount type=bind,source="$(pwd)"/tmp,target=/home/ep/var/tmp \
   acme/epdemo-hrp
```

Windows:

```
docker container run --name epdemo-hrp -p 8010:8000 -d ^
   --mount type=bind,source="%cd%"/var,target=/home/ep/var ^
   --mount type=bind,source="%cd%"/tmp,target=/home/ep/var/tmp ^
   acme/epdemo-hrp
```

An alternative to using bind mounts for storing the runtime data is using docker volumes. Such volumes will be stored in a special directory on the host running the container. The path to that directory depends on the installed docker client software, so please refer to the documentation of your docker client software in use.

Here is an example of the run command using a docker volume:

```
docker container run --name epdemo-hrp -p 8010:8000 -d -v ep-var:/home/ep/var acme/epdemo-hrp
```

This command uses a volume named `ep-var`. If no such volume already exists then it will be created. But there is a problem now: the volume does not contain our prepopulated `ep.conf`, and we do not want to bind mount it into the container, as this would end up in storing configuration changes in the mounted file and not in the volume.

To solve this, you need to change the Dockerfile for the image creation and add the following lines at the very end of `Dockerfile-ep-hrp`:

```
COPY ./ep.conf var/conf/ep.conf
VOLUME /home/ep/var
```

By defining the volume in the docker file all the content of the specified directory within the image will be copied to the volume. As our prepopulated `ep.conf` is located within `/home/ep/var/conf`, the file will now be available when starting the container with the

command above.

Note that the image now contains sensible information, e.g. the license key, passwords. See section Further considerations for further information on how to solve this.

5.4 Advanced docker file

For a better understanding, in all our previous examples we tried to keep the docker files as simple as possible. But if it comes to productive operation a more sophisticated docker image is needed. That file may still be customized to support different stages like test or pre-production. The following docker file may serve as a template for this purpose

File **docker/Dockerfile-ep-prod**

```
# docker file installing ep in alpine linux with a jre 17
FROM eclipse-temurin:17-jre-alpine
LABEL maintainer "support@acme.com"

# Setup EP_HOME -- useful when modifying the image in another Dockerfile
ARG EP_HOME=/home/ep
ENV EP_HOME=${EP_HOME}

#create user and group for running ep
ARG user=epserver
ARG group=epserver
ARG uid=1099
ARG gid=1099

# @enterprise is run with user 'epserver', uid = 1099
# If you bind mount a volume from the host or a data container,
# ensure that the user is permitted to read/write the mounted files and directories
RUN addgroup -g ${gid} ${group} \
  && adduser -h "$EP_HOME" -u ${uid} -G ${group} -s /bin/sh -D ${user}

USER ${user}

# set the working directory for all commands
WORKDIR ${EP_HOME}

ARG EP_VERSION=*

# mount the host directory containing the ep-setup file and start the installation
RUN --mount=target=/tmp/ep-files java -jar /tmp/ep-files/
     ep-setup-${EP_VERSION}.jar . default 0

ARG DB_DRIVER=h2-2.2.224

# copy the database driver into the lib directory of ep server
COPY ./${DB_DRIVER}.jar ./local/lib/${DB_DRIVER}.jar

# extract the application into appls-dir using a mount to the build context
```

```
RUN --mount=target=/tmp/ep-files unzip /tmp/ep-files/hr.zip -d appls/hr

# make the default port accessible
EXPOSE 8000

# define the volume for storing the runtime data of the server instance
VOLUME ${EP_HOME}/var

# start @enterprise
CMD ["sh", "base/epstart.sh"]
```

This file provides the following improvements:

- **Installation directory**: via argument `EP_HOME` you may change the directory in which *@enterprise* server will be installed
- **User and group**: via arguments `user`, `group`, `uid` and `gid` you can customize the names and ids of the user and group in which context the *@enterprise* server is run. Using an own user and group to run *@enterprise* server is essential to avoid running this process with root permissions.
- ***@enterprise* version**: to guarantee that the setup file of the intended *@enterprise* version will be used when building the image you can specify the version number. If no such file exists in your build context the build will fail.
  If no version information is specified and the build context contains exactly one file which name matches `ep-setup-*.jar` that file will be taken to build the image, otherwise the build will fail.
- **Database driver**: to avoid that you need to write a docker file for every single database driver you want to use you can customize this docker file via argument `DB_DRIVER`. So you can e.g. create an image for your test environment using one database product and use the very same docker file to create the image for your production system where different database product is used.
- **Volume**: the volume for the stored runtime data of your *@enterprise* server instance is defined

Here is an example of using this docker file to create an image using a driver for PostgreSQL:

```
docker image build -t groiss/epdemo-prod -f Dockerfile-ep-prod --build-arg EP_HOME=
/home/epdemo --build-arg DB_DRIVER="postgresql-42.5.0" .
```

5.5 Further considerations

- **Sensible information**: Sensible information like license keys, passwords or certificates should not be part of an image. This information should come from outside the image, e.g. by mapping appropriate files into the container.

- **Configuration files**: When running multiple containers of the same image there may be *@enterprise* configuration parts which are the same for all containers and some which are not. In that case you should consider to split up your configuration into multiple files so that the shared ones may be part of the image itself (most likely in directory `local/conf` of the server) whereas the container specific ones are only mounted into the container.

  In that case you need to set the environment variable `EP_CONFFILES` either in your build file or when starting the container. For further information about multiple configuration files please see section Multiple configuration files.

  Hint: When adding configuration files to your image keep in mind that those files should not hold sensible information.

- **External databases**: Until now, in our examples we always used H2 as embedded database which is only an option for a development or test system. When using other databases it may be the case that such a database is not already available when the *@enterprise* container is started - e.g. when the database itself is run in a container and both, database and *@enterprise*, are started as a group of containers. In this case the *@enterprise* server would not be started correctly, neither could the *@enterprise* server check if the database schema is already available or up-to-date nor could the corresponding initialization/upgrade be performed.

  To solve this the following *@enterprise* configuration parameters need to be set in `ep.conf` like:

  ```
  database.waitFor.seconds=5
  database.waitFor.count=20
  ```

  The specific values may be different in your environment depending on the time really needed by your database to become available.

- **Logging**: You should consider to activate parameter 'Log on Console' via `logger.logOnConsole=true` in your configuration file. In this case you can use the following docker command to observe the logging output of your *@enterprise* container:

  ```
  docker container logs <container-name> -f
  ```

6 Clustered @enterprise System

6.1 Overview and Principles of the Clustered Architecture

@*enterprise* can also be deployed as a clustered system. The aims of such an architecture are:

- increased scalability,
- increased availability,
- easier configuration,
- more flexible operation.

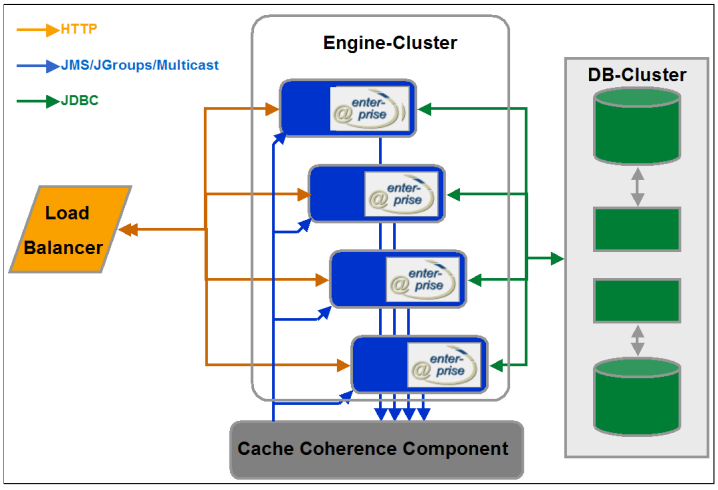

Figure 6.1: **Cluster Architecture**

Figure 6.1 shows the principal layout of such a cluster. The logical architecture consists of a set of @*enterprise* engines (termed "nodes") which access a common database and are operated in a peer to peer mode to a large extent.
A load balancing mechanism can be employed to ensure even load distribution within the cluster.
Consistency between the caches in the nodes is ensured by a cache coherence service.

While there are no single points of failure within the cluster nodes, we require the database to be available and scalable to an extent that imposes no bottlenecks for the rest of the system. In this section we will describe the basic configuration of such a clustered system without any external load balancer. A more complex configuration with an external load balancer will be presented in chapter @*enterprise* in a Load Balancing / Reverse Proxy Environment.

6.2 Cluster and Nodes

A node is running within a single Java Virtual Machine instance. In a typical production environment, there will be one node running on a single physical machine. In a development or test environment, more than one node could be running on one machine (without enhanced scalability and availability).
The cluster as a whole is represented by a single entry in the Cluster/Server section of the administration. Each node is identified by a Node-Id which must of course be unique within the cluster. Nodes can enter and leave the cluster at runtime. New nodes can be added to the cluster on the fly.

6.3 Configuring a clustered @enterprise System

Clustering of an @*enterprise* system will typically comprise the following actions:

- configuration of the underlying platforms in terms of hardware, operating system, network and database connectivity as well as JVM,
- installation of a single (nonclustered) @*enterprise* system,
- selection of the appropriate transport mechanism for the cache coherence service, its configuration and startup if necessary,
- distribution of the @*enterprise* installation directory to the nodes,
- adapting the @*enterprise* configuration (optionally storing the configuration within the database),
- starting the nodes.

Details for each of the steps can be found in the following sections.

6.3.1 Platform Configuration

The nodes of an @*enterprise* cluster can run on a heterogeneous platform as far as the hardware and operating system is concerned. While it is also possible to use different versions of the JVM/JDK it is strongly recommended to use the same principal version for each node. If your installation must use different versions, intense testing is strongly advisable.
The requirements for the minimal technical layout of the nodes do not differ from those for the layout of a single machine. A possible exception are the network interface requirements. It may be advisable to use different physical network interfaces and interconnections for client connections, database connections and possibly for the cache coherence service.

6.3.2 Installation of a nonclustered System

Just install a plain *@enterprise* system and make sure that it is working. Any cluster specific actions should be carried out afterward.

6.3.3 Adapting the *@enterprise* Configuration

Configuration: The following configuration entries are needed in a clustered node under *Configuration / Cluster* or *ep.conf*:

- **Clustering enabled - avw.cluster.activated:** Must be checked.
- **Server name - avw.servername:** Name of the server. Must be the same on each node of one cluster.
- **Node Id - avw.node.id:** Id of the cluster node. Must be unique within the cluster.
- **Performance factor - avw.node.perffactor:** Relative performance factor of the node. Depends largely on CPU power of the node. A node with a factor of 2 is expected to support twice the users of a node with factor 1. The load balancer makes use of the factor to distribute user sessions according to the relative power of the nodes.
- **Member of load balancing - avw.node.loadbalancing.member:** If set to *YES* (by default), the loadbalancing function for this node is active, i.e. the node is a potential target for clients which request loadbalanced sessions. On nodes which serve special purposes and should not receive logins from "ordinary" clients, this parameter should be unchecked.
- **Set Node Cookie - avw.cluster.setnodecookie:** Needed for load balancing with sticky local sessions in a proxy environment; not to be used for sticky but distributed sessions (see chapter *@enterprise* in a Load Balancing / Reverse Proxy Environment).
- **Coherence strategy - avw.dbcache.coherence.strategy:** Currently there is just one strategy supported: Notification. Do not confuse this with the client notification mechanism. Besides sharing the same name, they have nothing in common. In the future, other strategies might be provided as well.
- **Transport layer for Coherence - avw.dbcache.coherence.transport:** Choose the appropriate transport mechanism as described in section Transport Mechanisms for Cache Coherence Service.
- **Disallow logins after coherence error - avw.cluster.coherence.error.disallowslogin:** If this checkbox is activated, no logins are allowed anymore in case of a coherence error.

Ports: If you do run several nodes on one machine (e.g. for testing purposes), ensure that distinct network port numbers for the HTTP server, the HTTPS server and the RMI-mechanism are used.

Directories: If your nodes run on the same machine or access the same remote file systems, be sure to configure each of the nodes with distinct destinations for the variable parts of the node (by setting the variable area via the `EP_VAR` environment variable or the `ep.var` java system property as described in section File structure of stand-alone server).

Timers: Timers require special consideration in a cluster. There might be timers which should run on each node, and there might be timers that should only be running on one dedicated node of the cluster. The former timers must just be marked by checking the box *Run on each Node* on the timer edit form.
The latter ones must be marked by NOT checking the box and require special action. In a clustered system one of the nodes assumes responsibility for running the timers. Transparent failover is provided.
To enable this functionality, make sure that two timers are started on each node (c.f. Administration/Applications/Default/Timers):

- **HeartBeat:** Should be running on each of the nodes. Periodically writes a timestamp to the database. Used to monitor cluster nodes. During normal operation, there is exactly one update of a single row followed by a commit per heartbeat (and node). The heartbeat mechanism uses a dedicated database-connection when more than five database connections have been configured for the node, eliminating stalls due to finding a connection, as well as overhead due to frequently releasing and reacquiring the connection. Recommended periods are in the range from 3 to 10 seconds. Because of these short heartbeat intervals it is recommended to use a dedicated timer thread by assigning a unique thread-id (e.g. "heartbeat") to the timer. This avoids the possible delay of the heartbeat by other (longer-running) timers, thereby transferring the heartbeat info into the database as fast as possible. In order to avoid false positives, due to occasionally missed heartbeats, a tolerance time span can be specified by entering a single integer in the timer *arguments* field. The tolerance time span is measured in seconds, recommended values are at least triple the timer period.

- **ClusterCheck:** Should be running on each of the nodes. Periodically checks the health state of the cluster. There are two aspects to account for. First, if a node fails to update its timestamp, its state is set to "not running". Second, if none of the nodes runs the timers, which are started just once for the whole cluster, one node must assume this role. Appropriate timer run periods are in the range from 120 to 600 seconds. Again, we recommend to assign a unique thread-id (e.g. "clustercheck") to the timer. A *Clustercheck Tolerance Time* can be entered in the *arguments* field of the timer mask. It specifies the interval (in seconds) that needs to have elapsed since the last heartbeat of a node, before a switch of the cluster timers to another node takes place.

6.3.4 Optional synchronization of the configuration using the database

The configuration files of @*enterprise* (e.g. ep.conf) and of the applications (appl.prop) can be mirrored in the database to facilitate cluster-wide synchronization of common parameters in those files. We strongly recommend to have a working clustered system at hand before attempting to use this feature; if this is your first cluster, skip to section Transport Mechanisms for Cache Coherence Service.

To enable the configuration backing in the database, the following configuration steps should be performed:

1. Create an additional configuration file for each cluster node - e.g. *node_N.conf* (N is the Node id defined in configuration parameter *avw.node.id*) - which contains the node specific configuration parameters that should **not** be synchronized, especially the configuration parameter *avw.node.id* must be entered in each node-specific file! All other configuration parameters which should be synchronized between the cluster nodes must be stored in the cluster-wide *ep.conf*. Each parameter should either be placed in all node-specific configuration files or exclusively in the cluster-wide configuration file. Each node-specific file must have a unique name and should be stored just on the corresponding node.

   Example configuration of *node_N.conf* in a clustered environment where the nodes are on different machines in the network (typical in a production environment):

   ```
   #essential
   avw.node.id
   #optionally
   avw.node.perffactor
   avw.node.loadbalancing.member
   ```

   Example configuration of *node_N.conf* in a clustered environment where the nodes are on the same machine (e.g. test environment):

   ```
   #essential
   avw.node.id
   #optionally
   avw.node.perffactor
   avw.node.loadbalancing.member
   avw.formclassdir
   Httpd.tempDir
   logger.logfile
   logger.errorfile
   #if Jetty/standalone deployment is used
   httpd.port
   http.ip-address
   ssl.port
   ssl.ip-address
   httpd.admin.port
   httpd.admin.ip-address
   ```

2. Define a comma-separated sequence of paths to the appropriate configuration files via the environment variable *EP_CONFFILES* depending on your deployment scenarios:

   - before calling `epstart.bat` (standalone installation with embedded Jetty under Windows)
   - during service installation via `service\epservice.bat` and `service\environment.bat` (standalone installation with embedded Jetty as a Windows service)
   - before calling `epstart.sh` (standalone installation with embedded Jetty under Linux)

- in the service description file `service/epservice.service` (as systemctl service under Linux)
- in setenv.bat or setenv.sh in the bin directory of Tomcat (if deploying under Tomcat)
- at the appropriate location for your application server

Hint: The file *ep.conf* must be always the last file in the sequence. In this file all new parameters will be inserted when the configuration is changed (that is, new parameters will be in the cluster-wide scope).

Example configuration when calling epstart.bat or when using service\epservice.bat:

```
set "EP_LOCAL=C:\epwf\shared"
set "EP_VAR=C:\epfw\node1"
set "EP_CONFFILES=%EP_VAR%/conf/node_cn1.conf,%EP_LOCAL%/conf/ep.conf"
```

Example configuration when calling epstart.sh:

```
export EP_LOCAL=/epwf/shared
export EP_VAR=/epfw/node1
export EP_CONFFILES=$EP_VAR/conf/node_cn1.conf,$EP_LOCAL/conf/ep.conf
```

Application server under LINUX when deployed under context root /epwf

```
export EP_LOCAL_EPWF=/epwf/shared
export EP_VAR_EPWF=/epfw/node1
export EP_CONFFILES_EPWF=$EP_VAR_EPWF/conf/node_cn1.conf,\
$EP_LOCAL_EPWF/conf/ep.conf   (on one line)
```

3. Set parameter `ep.configuration.store.in.database=true` in *ep.conf*. This parameter is also available in section *Configuration/Other parameters*.

If the configuration steps have been performed successfully for each cluster node, the cluster is ready for use now. During node startup, the following steps are performed automatically:

- Read configuration files first
- Start the `DBConnPool` to get access to database
- Synchronize parameters between database and configuration files by means of the file date (maximum modified time and change date from header) and attribute "changedat" of the database table *avw_config*
- Reload configuration and continue startup

If a configuration parameter is changed via *@enterprise* administration GUI, the changes are written in the corresponding configuration file in the file system at the current cluster node and also in the database. If a configuration parameter is changed directly in a configuration file when a cluster node is running, the function *Reload configurations* in *@enterprise* administration under *Admin-Tasks/Server/Server control* must be performed to synchronize the configuration files and the database for this cluster node.

No automatic synchronization of configurations between cluster nodes is intended. Each and every single synchronization must be triggered manually via the function *Reload configurations* on mask "Server control" (see *System Administration Guide* of *@enterprise*), i.e. if changes in the configuration (with intended cluster-wide scope) have been made on cluster node 1, then on all other cluster nodes, the function *Reload configurations* must be performed manually to synchronize the configuration for those nodes.

Please note that changes to parameters needed by the synchronization mechanism itself (the parameter *ep.configuration.store.in.database* and the JDBC parameters (`database.driver.class` `database.url`, etc.)) might require additional manual synchronization steps on each cluster node.

Please do also keep in mind, that no conflict handling of the cluster wide configurations in the database is implemented. The last writer of such a configuration will win. Let us give an example: if the value of parameter "x" has been changed to "v1" on cluster node 1 and is subsequently and independently changed on cluster node 2 to "v2", the value "v2" for "x" will finally be stored in the database. If afterward function *Reload configurations* is performed on cluster node 1, value "v2" will be stored for parameter "x" in the configuration file of cluster node 1 (instead of the originally intended value "v1"). It is recommended practice to routinely check whether the configuration is up-to-date (by means of the mask "Server control" mask) before changing it.

As already mentioned, the configuration files for applications (*appl.prop* files) are subject to the same synchronization mechanism. All parameters in those files are considered to be of cluster-wide scope.

6.3.5 Transport Mechanisms for Cache Coherence Service

The cache coherence mechanisms task is to propagate cache relevant events within the cluster in order to keep the caches up-to-date. For the time being, the following event types are propagated:

- **Workitems:** Changes in the worklist (new items, finished items, ...)
- **Substitution:** Changes in substitutions of users (new substitute, start and end of substitution periods)
- **Seen Objects:** Items that are new to a user.

We provide the following choice of transport mechanisms to account for different needs of an installation:

- Unreliable Multicast via UDP
- Reliable Multicast via JGroups
- Java Message Service (JMS)

Unreliable Multicast via UDP

While this mechanism is easy to configure and poses virtually no overhead, it is recommended primarily just for development or test installations, due to possible loss of packets. An installation which uses dedicated physical network interfaces and interconnections for cache coherence service might also use unreliable multicast with rather good results, but one should be aware of the susceptibility to errors. This transport mechanism uses features available in the Java platform, no special deployment or startup is needed.
The following configuration parameters are needed
Configuration → Coherence
and must be identical on all cluster nodes. These parameters are available only, if *Standard Multicast* is used as *Transport layer for Coherence*:

- **Multicast-IP-Address:** Must be a valid multicast address. No two clusters should use the same multicast address. Be aware of other applications using multicast in your configuration. For specification and assignments of multicast addresses, refer to *https://datatracker.ietf.org/doc/html/rfc5771*. Monitoring of multicast packets is quite easy with tcpdump (`"tcpdump -i <interface> ip multicast"`).
- **Multicast IP Port:** Port to send and receive multicast packets. Must be available on the machine.
- **Multicast TTL:** Determines the scope of multicast packets on the network. For clustered systems with small "network diameter" this should be 1.
- **Buffersize (Bytes):** Size of reception buffer in bytes. Recommended value is at least 30000 Bytes.

When specifying these values, be aware of possible address space collisions with a multicast-based client notification service, cache coherence services of other clusters or with session distribution via Hazelcast.

Reliable Multicast via JGroups

JGroups is an open source communications library for reliable group communication. It is written in Java (www.jgroups.org). It is deployed in the *@enterprise* engine itself and needs no external processes running. It is started automatically. The library itself consists of a single Java archive named `jgroups-*.jar`; it is shipped with *@enterprise*.
The following configuration parameters are needed
Configuration → Coherence
and must be identical on all cluster nodes. These parameters are available only, when *JGroups* is used as *Transport layer for Coherence*:

- **Groupname:** JGroups has the notion of communication groups. A member node must state the groups it belongs to. Can be an arbitrary string, we recommend to use *epgroup* or to use the name of the server entry in the cluster.
- **Properties:** This parameter specifies the location of a configuration file in XML-syntax.

The recommended configuration is located in the `ep-impl-<rev>.jar` file at `jgroups/ccs.xml`. When this configuration needs to be changed, copy the `ccs.xml` entry from the jar-file to a `jgroups` directory under the `classes` directory and carry out the changes there.
Since the whole JGroups protocol stack is configured through it, it looks rather complicated. But in normal situations, just a handful of key parameters need to be changed. Such parameters are clearly marked in the *ccs.xml* file. The parts of the configuration to be changed are the multicast IP address `mcast_addr`, the multicast port number `mcast_port`, the time to live `ip_ttl` and the `bind_addr`. For the multicast address and multicast port we refer to the previous section about unreliable multicast, for the "time to live" we recommend either 1 as the packets should only reach the other *@enterprise* nodes which are placed in the network vicinity. If network components are in between the nodes of the cluster, it might be necessary to increase this value to 32. The `bind_addr` is the address of the interface that should be used for the multicast communication. Be sure to change it from `localhost`.
In case of doubt, consult your local network administrator. Please avoid any interference within *@enterprise* (e.g. with RMI client notification service with multicast or with other cluster instances) when selecting multicast parameters.
The other properties in the file should not be changed without sound knowledge about JGroups.

Java Message Service (JMS)

The usage of JMS for the transport of cache coherence messages can be characterized as follows. The publish subscribe paradigm is used. Per node there is one subscriber and one publisher. All nodes subscribe to the same topic. No message selectors are used. We use non-persistent, auto-acknowledged, non-transacted messages and nondurable subscribers with `ObjectMessages`.
JMS does not run within an *@enterprise* JVM, it must be configured and started separately. Apache ActiveMQ (*http://activemq.apache.org*) meets all requirements and is known to be reliable, but virtually any JMS implementation should be suitable. Hints for configuring a particular JMS might be obtained from the *@enterprise* support.
The following configuration parameters are needed under
Configuration → *Coherence*
and must be identical on all cluster nodes. These parameters are available only, when *JMS* is used as *Transportlayer for Coherence*:

- **JMS Provider URL:** The URL name of the JMS provider. For ActiveMQ this is something like `tcp://<jmshost>:61616?wireFormat.maxInactivityDuration=10000`.
- **JMS ContextFactory:** Name of the Java class for constructing the JNDI-Context. For ActiveMQ this is `org.apache.activemq.jndi.ActiveMQInitialContextFactory`.
- **JMS TopicConnectionFactory:** Java class name for the topic factory of the JMS provider. For ActiveMQ this is `ConnectionFactory`.
- **JMS Topic:** The name of the topic used for communication. Such topics must typically be created within an JMS provider by the administrator. For ActiveMQ this can also be a dynamic topic like `dynamicTopics/ep11`.

- **JMS Time to Live (ms):** The JMS provider is free to discard messages which are older than this timespan. Should be in the range from 30 to 120 seconds. Unsynchronized clocks on participating systems might inhibit proper message transfer.
- **JMS Username:** Name of the user which is utilized for communication with the JMS provider. If this parameter is left empty, an anonymous connection is established. User administration is specific for each JMS provider.
- **JMS Password:** Password for the JMS user.

Hints for ActiveMQ: [1]

- The JMS broker can be started via `activemq start` in the `activemq/bin` directory (be sure to set an appropriate value for `JAVA_HOME`).
- The minimal set of libraries for a client is `activemq-client-*.jar, hawbuf-*.jar`, both are located in the `activemq/lib` directory.
  Put those two files in the *@enterprise* classpath at each node, preferably in the `local/lib` directory.
- To allow for proper `ObjectMessage` deserialization, set the java property
  `-Dorg.apache.activemq.SERIALIZABLE_PACKAGES=*`
  at each node, preferably via the `local/localjavaargs` file.

More than one cluster can use a JMS provider, if you make sure that the names of the topics are distinct for each cluster. Also do not use the same topic name for cluster coherence via JMS and for RMI client notification via JMS if you are using the same physical provider for both purposes.

6.4 Operation of a clustered system

6.4.1 Monitoring

A cluster health monitor which displays the state for each of the nodes can be accessed via *Admin-Tasks / Cluster / Clustermonitor*.
The fields displayed are:

- **Hostname:** Name of the cluster.
- **Node-Id:** Id of the node.
- **Start Time:** Time of startup of this node.
- **Last HeartBeat:** Timestamp of last heartbeat made by this node.
- **Running:** Shows, whether the node is running.
- **ClusterTimers:** Shows, whether the node is the one which runs the cluster timers.

[1] Since we do not ship ActiveMQ with *@enterprise*, the instructions may need to be adapted for specific versions; we used version 5.18.0.

- **Load:** Current number of connected users.
- **Performance Factor:** The performance factor of the node.
- **Load Coefficient:** The current load coefficient (number of users divided by performance factor).
- **Load Balanced:** Shows, whether the node is a member of the loadbalancing nodes (see section Internal Load Balancing).
- **Logins enabled:** Shows, whether new logins are allowed on the current node. Logins can be enabled/disabled with the toolbar-function *Disable/Enable Login*.
- **Current Session:** Shows if current sessions should be kept, renewed or aborted. At startup this is always set to "keep".
- **Successor Nodes:** The id of the successor node where the clients of the "switched off" node should be logged in, when login is restricted (see column *Logins enabled*) and current session should not be kept. At server startup this column is always empty.

Hint: *Current Session* and *Successor Nodes* are used by *@enterprise* RMI Clients only!

6.4.2 Internal Load Balancing

Principle We provide some basic loadbalancing functionality without the need to deploy an external loadbalancer / reverse proxy. A client which wants to obtain a load balanced session should first connect to a special URL on an arbitrary running cluster node.
There, the client will be redirected to the least loaded node (HTTP(S)-Client) or can obtain appropriate initial URLs of this node (RMI-Client).
Please note that this mechanism does not necessarily imply the use of a dedicated front-end load balancing system. Details for a more complex configuration with the use of a dedicated front-end load balancer can be found in chapter *@enterprise* in a Load Balancing / Reverse Proxy Environment.

HTTP(S)-Clients The URL for getting a load balanced session for an HTTP(S) Client is:

```
http[s]://<host>:<port>/<context-root>/
servlet.method/com.groiss.cluster.ClusterSupport.redirect
```

The client will be redirected immediately to the server with the lightest load.

RMI-Clients Use the same mechanism as mentioned above. A client should open an URL-Connection to:

```
http://<host>:<port>/<context-root>/
servlet.method/com.groiss.cluster.ClusterSupport.getRedirectAddresses
```

Three URLs are returned, each in a separate line. The URLs can be used by the client to obtain an appropriate session to the node. The first URL is the one for HTTP clients, the second one is the URL for RMI clients, the third one is the URL for HTTPS clients. This data can be used by the client to obtain a session to that node.

6.4.3 Further cluster aspects

Other aspects in the cluster context which are worth mentioning are:

- Event handler execution: Event handlers (`sync/wait`) are executed on the node where the event has been raised.
- Timer change propagation: In the configuration section *Other parameters* the configuration parameter *ep.timerentry.change.check.seconds* can be used to periodically check for changes in timer entries. This enables changes in timer specifications to be propagated throughout the cluster.

7 @enterprise in a Load Balancing / Reverse Proxy Environment

We will discuss some key aspects of deploying an *@enterprise* clustered system behind a load balancing / reverse proxy.

7.1 Basic Constellation

A cluster consists of several *@enterprise* nodes; each with its own unique node id. Each node provides HTTP(S) services to users addressed via a host name / port combination, respectively an IP-address / port combination. All nodes together comprise the cluster (which is - somewhat confusingly - for historical reasons and called the "Server").
Installing a reverse proxy between the nodes and the clients strives to fulfil three main purposes:

- Providing a node transparent view of the *@enterprise* system
- Load balancing
- SSL termination

7.2 Main Technical Considerations

7.2.1 HTTP Session Binding (Sticky Sessions)

In an *@enterprise* cluster, requests for a client in the scope of a session need to be bound to an arbitrary, but particular node. So *@enterprise* needs a proxy that supports session binding (also called sticky sessions or persistent sessions).

A session in *@enterprise* is conveyed in the form of a session cookie with the name `<serverid>_EPSESSIONID`. In order to provide session binding to nodes, a second cookie is issued. This routing cookie is named `<serverid>_EPNODEID`, its value is the node-id of the node the session is bound to.
The technical mechanisms for this routing cookie differ depending on whether session failover is being used or not. They will be elaborated on in the next section.

7.2.2 HTTP Session Failover

In @*enterprise* there are two modes of operation concerning session failover:

- **Node local sessions:**
  they are stored in the file system of the local node (the value of property `httpd.jetty.session.store` in section *other parameters* of the configuration must be set to `File`). No failover is provided. No session data is distributed within the nodes. When a node fails, the proxy has to detect this state and will reroute the request to use another (working) node. Since the HTTP session is not known on the new target node, the user will have to log-in again to this node.

  The node routing cookie for session binding must be switched **on** via the parameter `avw.cluster.setnodecookie` in the *Cluster* section of the configuration.

  In @*enterprise*, the routing cookie is set at login time and deleted at logout time. The proxy must be configured to honor the node-id cookie inasmuch that incoming requests providing the cookie must be routed to the corresponding node. The routing cookie always has the name `<serverid>_EPNODEID`.

- **Cluster wide sessions:**
  are implemented by using a distributed computing framework (`Hazelcast`). Sessions are kept in memory and distributed within the cluster nodes. Changes in sessions are propagated within the cluster, there is always a primary location for a session as well as a backup location. Single node failures do not result in loss of sessions. Loss of one copy of session data results in redistribution of sessions, so there are again two copies of the sessions available. Node failure still has to be detected by the proxy and also rerouting of the session does take place, but the session is still the same, so there is no need for the users to re-login again.

  In this case the node routing cookie is created and dynamically changed by the proxy. The generation of an internal routing cookie must be switched **off** via the parameter `avw.cluster.setnodecookie` in the *Cluster* section of the configuration. The routing cookie will not be set by @*enterprise* but by the proxy, at logout time the cookie is deleted by @*enterprise*. Also in this case, the cookie always has the name `<serverid>_EPNODEID`.

  There are some notable aspects of cluster-wide sessions:

  - Technical environment:
    The cluster wide session mechanism is integrated in the embedded Jetty server. It will not work in other environments like Tomcat or other application servers. Hazelcast runs within the JVM of @*enterprise*, no additional service needs to be started, appropriate memory must be provided within the JVM.
  - Configuration:
    To use this mechanism, appropriate configuration must be applied to @*enterprise* to the Hazelcast system itself, and to the proxy. Those aspects are dealt with throughout the rest of this chapter.
  - Nonserializable attributes:
    Attributes that are transient cannot be transported between nodes. Such attributes

are filtered out before handling them over to the distribution mechanism. They never appear at other nodes.

- Changes of session attributes:
  Propagation of changes of session data in the cluster relies on calls to `httpSession.setAttribute(name,value)` and to `httpSession.removeAttribute(name)`. If the state of an attribute changed, propagation takes places only after another call to `httpSession.setAttribute(name,value)`.
- Complete session loss:
  If all nodes of a cluster are shut down, all sessions are lost, there is no persistence mechanism provided.

Configuration of Hazelcast for session failover

To use cluster wide sessions, the parameter `httpd.jetty.session.store` in section *other parameters* must be set to `Hazelcast`.

The default Hazelcast configuration is to run within the JVM and requires no separate start. The needed components are shipped with *@enterprise*.
We also need to switch the parameter `avw.cluster.setnodecookie` **off** and set the parameter `httpd.jetty.behindproxy` as described above.

The Hazelcast configuration is in the `ep-impl-<rev>.jar` file at path `hazelcast/sessionstore.xml`, this file is based on the default Hazelcast configuration and has a number of configuration hints at its beginning. If this configuration needs to be changed, copy the entry `sessionstore.xml` from the jar file to a `hazelcast` directory under the `classes` directory and carry out the changes there.

Pay special attention to network addresses and ports and configure your network infrastructure appropriately. While *@enterprise* uses the `serverid` to distinguish different clusters in the same network, it is wise to also choose network properties that would not interfere with each other or with other multicast channels like JGroups.

7.2.3 Node Election at initial Session Creation

At the first contact between a client and the *@enterprise* cluster, the cookie bearing the node-id has not been set. So the choice of the initial node is somewhat arbitrary. The proxy should be configured to use some sensible strategy. By using a special login URL (see section Special Functions), the *@enterprise* load balancing mechanism - which is based on a combination of node weight and user session count - can also be used for initial node election.

7.2.4 SSL Termination in Proxy

In a configuration where all traffic between the proxy and the clients is carried out via SSL/TLS, a function of the proxy is to terminate the SSL traffic and to use plain HTTP to connect to the cluster nodes. This diminishes the SSL processing load on the *@enterprise*

nodes.

Since this would be within the perimeter of a central network, unencrypted data traffic should not be a security issue there. Should SSL termination at the proxy really not be allowed, a different configuration is needed, since the cookies are also encrypted in such a scenario and are therefore not accessible for the proxy for node routing purposes.

7.2.5 Transparent View for the Clients

To present a uniform and node independent transparent address for the clients, the *@enterprise* server object must be configured accordingly. The "official" address of the clustered system must be set. In particular, the combination of protocol (HTTP or HTTPS), host name and port (which will be used by the client to contact the proxy) must be provided via *Administration/Admin-Tasks/Cluster/Servers*.

If the nodes are configured to use dedicated admin connectors, the proxy configuration as well as the server info must be augmented accordingly.

7.2.6 HTTP Header Transformation by the Proxy

In order to provide the nodes with appropriate data about the technical nature of each request, the headers of the HTTP requests must be enriched by the proxy. Two additional headers are important: `X-Forwarded-For` and `X-Forwarded-Proto`. The first one bears the originating client address, the second one should be set to `https`, when the proxy uses SSL session termination and the incoming connection was made via SSL.

7.2.7 Configuration considerations for *@enterprise*

An additional parameter `httpd.jetty.behindproxy` must be set in section *Other parameters* in the configuration; it accounts for correct interpretation of the `X-Forwarded-*` headers. For the correct setting of the `avw.cluster.setnodecookie` parameter, we refer to section HTTP Session Failover.

7.2.8 Special Functions

Some special functions for a proxied configuration are provided via explicit URLs. The prefix for all those URLs is:

```
protocol://proxy:port/ctxroot/servlet.method/com.groiss.cluster.ClusterSupport.
```

The following functions are available:

- `redirect`: Uses the *@enterprise* load balancing mechanism to contact the least stressed node (sets the node routing cookie).
- `redirect?nodeid=<nodeid>`: Explicitly sets the node routing cookie to a dedicated node.
- `unbindAndRedirect`: Removes the node routing cookie, then a redirect redirection will occur.

- `getClientInfo`: For troubleshooting purposes, all details about the current request are presented.

7.3 Example Configurations

In the following, we outline the configuration of an @*enterprise* system with HAProxy (using version 2.6.12 available via *http://www.haproxy.org/*) as a reverse load balancing proxy.

We will present a basic configuration with HAProxy. First, we describe the constellation of the @*enterprise* nodes. Then we describe the preparation of HAProxy and present a common configuration followed by a summary of the differences between local and distributed session modes.
The configuration should be seen just as a (working) starting point; there are literally dozens of proxy configuration options and tuning parameters which might need to be adjusted to derive a production ready configuration variant.

7.3.1 @*enterprise* Constellation

We will use an @*enterprise* clustered system consisting of three nodes. Each node has two connectors: a HTTP user connector and an admin-connector which uses SSL (but is terminated by the proxy).

NodeId	Host name	User port	Admin port
Node_1	n1.acme.com	8001	8011
Node_2	n2.acme.com	8002	8012
Node_3	n3.acme.com	8003	8013

We assume, that the proxy will be reachable via:

ServerId	Host name	User port	Admin port
epcluster	enterprisewf.acme.com	80	443

So we have to set the @*enterprise* server object (c.f. subsection Transparent View for the Clients):

- *Protocol:* HTTP
- *Hostname:* enterprisewf.acme.com
- *HTTP(S) port:* 80
- *Administrative protocol:* HTTPS
- *Administrative IP port:* 443

7.3.2 Preparation: Proxy Building and SSL Aspects

We used HAProxy version 2.6.12 for the example:

- *Compilation:* The options for the make command will have to be adjusted according to the platform in use. For a reasonable modern Linux system, the following actions might be appropriate:

```
tar zxvf haproxy-2.6.12.tar.gz
cd haproxy-2.6.12
make TARGET=linux-glibc USE_OPENSSL=1 USE_SYSTEMD=1
```

- *SSL configuration:* The following steps can be used to provide the proxy with a self signed server certificate:

```
# generate a private key
openssl genrsa -des3 -out privkey.pem 2048
# generate certification request
openssl req -new -key privkey.pem -out cert.csr
# sign the certificate request
openssl x509 -req -days 10000 -in cert.csr -signkey privkey.pem -out cacert.pem
# the following two commands are needed, if HAProxy startup should
# not ask for manual input of the passphrase
cp privkey.pem privkey_passphrase.pem
openssl rsa -in privkey_passphrase.pem -out privkey.pem
# build a certification chain
cat cacert.pem privkey.pem > cacertprivkey.pem
```

  We assume that the `cacertprivkey.pem` is in directory `/var/haproxy/cnf`.

- Installation and start of the HAProxy service: depends on your flavor of operating system, follow the steps outlined in the HAProxy documentation.

7.3.3 Common proxy configuration

In the following, we provide a commented configuration file for *HAProxy* for the installation outlined in section @*enterprise* Constellation on the host *enterprisewf.acme.com*.
Most notably, the configuration of the cookie handling mechanism is conditional depending on the value of an environment variable `EP_SESSION_MODE`. This variable has to be set at the start of HAProxy. If the variable is set to `distributed`, then a cookie handling appropriate for distributed sessions will be used. For any other value of the variable, we use a form of cookie handling suitable for local sessions.
Since you will be using one mode only, you can refrain from using the environment variable and just plainly put the appropriate `cookie` line into the `haproxy.cfg`.

```
# example configuration of haproxy 2.6.X
# as reverse load balancing proxy in front of a cluster.
# session mode depends on environment variable EP_SESSION_MODE
global
   description epcluster
   # run in background
   daemon
```

```
   # user and group ids for execution environment
   #user HAProxy
   #group HAProxy
   # empty unwritable jail dir for chroot
   # chroot some_dir
   # max number of connections per process
   maxconn 1024
   # logging definition syslog or systemd
   # syslog
   log 127.0.0.1 local1 notice
   # systemd
   # /dev/log local1 notice
   # base directories for ssl stuff
   crt-base /var/haproxy/cnf
   ca-base /var/haproxy/cnf

defaults
   # operating mode (layer 7 inspection)
   mode http
   # continuously update statistics; enable for testing; small performance impact
   option contstats
   # health checks are logged only when state of server changes;
   # can be enabled in production, too
   option log-health-checks
   # timeouts; should be adjusted to match network and backend configuration
   timeout connect 5s
   timeout client 30s
   timeout server 30s
   # use a long timeout for bidirectional tunnel traffic (e.g. websockets)
   timeout tunnel 1h
   # use default mode
   option http-keep-alive
   # add x-forwarded-for header
   option forwardfor
   # use logging definitions from global section
   log global
   # uses cookie <serverid>_EPNODEID for session affinity (sticky sessions)
   # depends on environment variable EP_SESSION_MODE
   .if streq("${EP_SESSION_MODE}","distributed")
      # use with distibuted sessions / hazelcast
      cookie epcluster_EPNODEID insert preserve
   .else
      # use with node local sessions
      cookie epcluster_EPNODEID nocache
   .endif

# listens on port 80 for incoming http requests (user connector)
frontend http-in
   description user-port 80
   bind *:80
   default_backend ep-workers

# listens on port 443 for incoming https requests (admin-connector)
```

```
frontend https-in
   description user-port ssl 443
   # use all network interfaces ;
   # the pem file is a concatenation of cacert and private key
   bind *:443 ssl crt cacertprivkey.pem
   # add  X-Forwarded-Proto header to mark request as secure for backend
   http-request add-header X-Forwarded-Proto https
   http-request set-header X-SSL %[ssl_fc]

   # all requests use the following backend (the admin-connector)
   default_backend ep-workers-admin

# the nodes (user-connectors)
backend ep-workers
   description nodes (user connectors)
   # use the node with the fewest connections
   balance leastconn
   # allow for redispatching a request; if a designated (via the cookie) server is down
   option redispatch
   # retries must be nonzero for redispatch to work
   retries 3
   # default options for all servers of this backend
   # check health via tcp; weight for loadbalancing
   # httpchk may be more adequate;
   #option httpchk GET /wf/servlet.method/com.groiss.cluster.ClusterSupport.getNodeState
   #http-check disable-on-404
   #http-check send-state
   default-server inter 10s weight 10
   # for each node a line must be inserted:
   # might be better to use ip-Addresses instead of hostnames
   # <Nodeid> <ip:user-port> cookie <Nodeid> check
   server Node_1 n1.acme.com:8001 cookie Node_1 check
   server Node_2 n2.acme.com:8002 cookie Node_2 check
   server Node_3 n3.acme.com:8003 cookie Node_3 check

# the nodes (admin-connectors)
backend ep-workers-admin
   description nodes (admin connectors)
   balance leastconn
   option redispatch
   retries 3
   default-server inter 10s weight 10
   # <Nodeid> <ip:admin-port> cookie <Nodeid> check
   # might be better to use ip-Addresses instead of hostnames
   server Node_1 n1.acme.com:8011 cookie Node_1 check
   server Node_2 n2.acme.com:8012 cookie Node_2 check
   server Node_3 n3.acme.com:8013 cookie Node_3 check

# admin connector for HAProxy console
listen admin
   description Administrative Overview
   # user port 10001 on all interfaces
   bind *:10001
```

```
stats enable
# default uri is:
# stats uri /haproxy?stats
stats uri /
stats realm HAProxy\ wfm \ Statistics
stats show-legends
# username:password
stats auth  admin:admin
stats admin if TRUE
```

7.3.4 Session Modes

The following table summarizes the commonalities and differences of the configuration between the session modes:

	Session mode	
Aspect	local	distributed
avw.cluster.setnodecookie	on	off
httpd.jetty.behindproxy	on	
httpd.jetty.session.store	File	Hazelcast
Hazelcast config	no	yes
HAProxy cookie name	cookie <serverid>_EPNODEID	
HAProxy cookie options	nocache	insert preserve

7.4 Operational Aspects of HAProxy

The *HAProxy* server process can be started via `haproxy -f <configfile>`, but of course it should be integrated into the startup sequence of your server. A brief dashboard of the current state of the cluster according to the *haxproy* server can be obtained via `http://enterprisewf.acme.com:10001`.
As default, HAProxy uses plain TCP health checks. As a consequence, a node appears to be available as soon as it accepts connections at the port of the HTTP connector. Since at this point in time, the node may still be starting up (e.g. loading the worklist cache), real operational readiness may be in effect somewhat later. Instead of the TCP check an HTTP check can be specified with the `option httpchk` in the HAProxy config file. Put the following config options into each backend definition:

```
option httpchk GET /wf/servlet.method/com.groiss.cluster.ClusterSupport.getNodeState
   http-check disable-on-404
   http-check send-state
```

Adjust the `/wf` prefix according to your context root. The lightweight `getNodeState` method returns the following states:

- 200: returned if node is loadbalanced and logins are enabled. HAProxy will mark this node as "UP/green".

- 404: if the node is loadbalanced, but logins are disabled and sessions should not be evicted. This means that HAProxy will mark this node as "NOLB/blue", so it will disable the node for new connections but will continue to serve requests on already existing persistent connections.
- 403: this is returned when the node is not loadbalanced; either because it is configured in this way statically, or because loadbalancing is set to `auto`, which depends on the state of the worklist cache. HAProxy will mark this node as "DOWN/red".

When using Hazelcast for session failover, there are some administrative functions available via the following URLs to gain some insight into the state of Hazelcast.

- `com.groiss.cluster.ClusterSupport.getHazelcastState`:
  displays general information about the Hazelcast subsystem. Parameter `showConfig=true` can be used to display the Hazelcast configuration, parameter `showDetails=true` can be used to display additional details for each session.
- `com.groiss.cluster.ClusterSupport.getHazelcastSessionInfo?sessionid=<sessionid>`:
  displays details about the session.

Hazelcast also offers an administrative console which can be used freely for clusters with no more than two nodes. Accessing larger clusters with this console requires a commercial license.

8 @enterprise and Datasources

This chapter describes the configuration of datasources in *@enterprise* and gives example configurations for *Tomcat* and *Jetty 6.1*.
Before version 8.0, *@enterprise* could use the traditional method to acquire connections to the database via the DriverManager. From version 8.0 and onward, datasources are an alternative way to obtain database connections.

8.1 Configuration of a Datasource in @enterprise

To use a datasource in *@enterprise*, the JNDI-path of the datasource must be specified instead of the JDBC-URL. Instead of e.g.

```
jdbc.derby://localhost:1527/ep;create=true
```

use something like

```
jdbc/DerbyDB
```

If the datasource path starts with './', it will be looked up in the initial JNDI context (without the './' prefix). If not, it will be looked up in the 'java:/comp/env/' subcontext of the initial JNDI context. When using the datasource it is not needed to provide a JDBC-driver or to fill in the following configuration items:

- Database Userid
- Database Password

The other database related configuration items are still needed and used.

8.2 Configuration of a Datasource in Tomcat

This section describes how Tomcat can be configured:

1. Put the JAR-file of the JDBC-driver into the *lib* directory of Tomcat.
2. Deploy the *@enterprise* WAR-file (e.g. using ep11 as the contextpath)
3. Go to *../conf/*<*service*>/<*host*> directory and put a <contextpath>.xml file there.

- *<service>:* The name of the Tomcat service (as in the service element in the ../conf/server.xml file); usually Catalina
- *<host>:* The name of the Tomcat host (as in the host element in the ../conf/server.xml file); usually localhost
- *<contextpath>:* The contextpath where *@enterprise* is deployed

So, you would end up with a file named ../conf/Catalina/localhost/ep11.xml. In this file specify the datasource as a resource within the context element:

```
<Context>
   <Resource
      name="jdbc/DerbyDB"
      auth="Container"
      type="javax.sql.DataSource"
      factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory"
      maxActive="12"
      username="derby"
      password="derby"
      driverClassName="org.apache.derby.jdbc.ClientDriver"
      url="jdbc:derby://localhost:1527/ep;create=true"
   />
</Context>
```

The value of the *name* attribute must match the path of the datasource in the *@enterprise* configuration file. The value of attribute *factory* can be also `org.apache.tomcat.dbcp.dbcp2.BasicDataSourceFactory` depending on your Tomcat version.

Details for the other parameters can be found in the Tomcat documentation.

4. The following step may not be needed. Include the reference to the resource in the *web.xml* descriptor of *@enterprise* application:

```
<resource-ref>
      <description>DB Connection</description>
      <res-ref-name>jdbc/DerbyDB</res-ref-name>
      <res-type>javax.sql.DataSource</res-type>
      <res-auth>Container</res-auth>
</resource-ref>
```

The content of the *resource-ref-name* element must match the path of the datasource in the *@enterprise* configuration file.

5. Restart Tomcat, start the *@enterprise* application and begin to setup *@enterprise*.

8.3 Configuration of a Datasource in Jetty 6.1

This section describes the configuration of *@enterprise* running as web-application in a jetty-installation (*@enterprise* does not start jetty as embedded web-server!):

1. Create a *myjetty.xml* file that activates the needed jetty-plus features. Using the *jetty.xml* and *jetty-plus.xml* files as model. Add the following lines to the server configuration element:

```
<Array id="plusConfig" type="java.lang.String">
       <Item>org.mortbay.jetty.webapp.WebInfConfiguration</Item>
       <Item>org.mortbay.jetty.plus.webapp.EnvConfiguration</Item>
       <Item>org.mortbay.jetty.plus.webapp.Configuration</Item>
       <Item>org.mortbay.jetty.webapp.JettyWebXmlConfiguration</Item>
       <Item>org.mortbay.jetty.webapp.TagLibConfiguration</Item>
</Array>
```

   Add the configuration classes also to the *addLifeCycle* call element:

```
<Call name="addLifeCycle">
 <Arg>
  <New class="org.mortbay.jetty.deployer.WebAppDeployer">
   <Set name="contexts"><Ref id="Contexts"/></Set>
   <Set name="webAppDir">
    <SystemProperty name="jetty.home" default="."/>/webapps
   </Set>
   <Set name="parentLoaderPriority">false</Set>
   <Set name="extract">true</Set>
   <Set name="allowDuplicates">false</Set>
   <Set name="defaultsDescriptor">
    <SystemProperty name="jetty.home" default="."/>/etc/webdefault.xml
   </Set>
   <Set name="configurationClasses"><Ref id="plusConfig"/></Set>
  </New>
 </Arg>
</Call>
```

2. Uncompress the *@enterprise* WAR-file (e.g. using ep11 as contextpath).

3. Put the JAR-file of the JDBC-driver into the *lib* directory of the web-application.

4. Go to the *../webapps/ep11/WEB-INF* directory and put a *jetty-env.xml* file there. In this file specify the datasource as a resource within a context element:

```
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
   <New id="DerbyDB" class="org.mortbay.jetty.plus.naming.Resource">
    <Arg>jdbc/DerbyDB</Arg>
    <Arg>
     <New class="org.apache.derby.jdbc.ClientDataSource">
                 <Set name="databaseName">ep</Set>
                 <Set name="portNumber">1527</Set>
                 <Set name="serverName">localhost</Set>
                 <Set name="user">derby</Set>
                 <Set name="password">derby</Set>
     </New>
    </Arg>
   </New>
</Configure>
```

The value of the first *Arg* element must match the path of the datasource in the @*enterprise* configuration file.

5. Restart Jetty with following parameter and begin to setup @*enterprise*:

```
java -jar start.jar etc/myjetty.xml
```

8.4 Considerations for pooled Datasources

@*enterprise* still uses its own connection pool, even when the datasource is a pooled one. We have better control over the connection this way, and can provide all features of the @*enterprise* pool itself (session environment, automatic reconnect, ...).
This strategy imposes two requirements for a pooled datasource:

- It should never expect to get the connection back or destroy connections in use. In a DBCP connection pool, this can be implemented via

  ```
  removeAbandoned="false"
  ```

  In a Weblogic pooled datasource this can be achieved via disabling the "Inactive Connection Timeout" feature by setting it to 0.

- The pool size should be large enough to provide the max. number of connections specified in the @*enterprise* configuration (see chapter Configuration) increased by at least 2 (for internal connections used by the engine itself).

9 OAuth 2.0 authentication

@*enterprise* can make use of OAuth 2.0 authentication for the reception and the transmission of e-mails.

In *OAuth* terminology, @*enterprise* is a confidential web client that acts in the name of the owner of the mailbox (mail account). For each installation, @*enterprise* must be registered once as a client against the *OAuth* provider. Logically distinct installations (like integration and production) must be registered as distinct clients.

After client registration, a corresponding Authorizer must be properly configured in @*enterprise*.

Then, the initial consent can be requested via the @*enterprise* admin GUI. The *OAuth Authorization Code* grant flow is being used for this purpose. The resource owner must log in once via the browser and give the consent. After the initial consent has been granted to @*enterprise*, the OAuth credentials (tokens) are kept current via the @*enterprise TokenRefresh* timer without any further user interaction (as long as the grant is still valid).

For the authorizer to be used for access to a mailbox, it must be selected in the mailbox configuration.

To use an authorizer for sending emails, its id must be entered in the *communication* section of the @*enterprise* configuration. While the underlying principles remain the same regardless of the specific authorization provider being used, the details can differ significantly.
In the following section we will go into the details for Google (Gmail) and Microsoft Azure (Office365).

9.1 Specific Configuration for Google/Gmail

9.1.1 Client registration

- Log in to *console.cloud.google.com*
- Create a new project or use an existing one
- Go to *APIs and Services/Library*

  - Select the Gmail API
  - Enable it

- Go to *APIs and Services/OAuth consent screen*
  - Select the appropriate User Type
    - For testing purposes, use *External*
    - For production, use *Internal*
  - Enter an *appname*, a *user support email address* and *developer contact information*
  - Add a scope: *https://mail.google.com/*
  - Add a test user: add the email address(es) you want to enable

- Go to *APIs and Services/Credentials*
  - Create credentials
  - Select *OAuth client ID*
  - Select *Web application*
  - Add an *Authorized redirect URI*
    - *<protocol>://<host>:<port>/<context>/servlet.method/com.groiss.auth.oauth2.OAuth.codeCallBack*
      - Where *<protocol>*, *<host>*, *<port>* and *<context>* must be changed according to your *@enterprise* installation. This URL is not public, it must be a *locally resolvable* URI where the browser will be redirected to.
  - Be sure to save the Client ID and the Client Secret!

9.1.2 Authorizer configuration for Google/Gmail

- In the *@enterprise* Admin Gui, navigate to *Admin tasks/Communication/Authorizers*

  NOTE: To automatically populate the fields in the authorizer use the question-mark icon near the *Authorization URI* field. Be sure to change the *<variables>* according to your specific installation.

- To manually create a new Authorizer please enter the following data:
  - Id:
    - An arbitrary id
  - Authorization URI:
    - *https://accounts.google.com/o/oauth2/auth*
  - Token URI:
    - *https://oauth2.googleapis.com/token*
  - Revocation URI (can be left empty):

        * *https://oauth2.googleapis.com/revoke*
    - Scopes:
        * *https://mail.google.com/*
    - Redirect URI:
        * *<protocol>://<host>:<port>/<context>/servlet.method/com.groiss.auth.oauth2.OAuth.codeCallBack*
            · Where *<protocol>*, *<host>*, *<port>* and *<context>* must be changed according to your @*enterprise* installation. This URL is not public, it must be a *locally resolvable* URI where the browser will be redirected to.
    - Client Id:
        * The client id obtained at client registration
    - Client secret:
        * The client secret obtained at client registration
    - Ask for offline access:
        * Check it
    - Use PKCE:
        * Check it
    - Additional properties: depends on intended usage and can be combined:
        * Properties for IMAP:
            · mail.imap.ssl.enable=true
            · mail.imaps.auth.login.disable=true
            · mail.imaps.auth.mechanisms=XOAUTH2
            · mail.imaps.auth.plain.disable=true
        * Properties for SMTP:
            · mail.smtp.starttls.enable=true
            · mail.smtp.auth.mechanisms=XOAUTH2
            · mail.smtp.auth.login.disable=true
            · mail.smtp.auth.plain.disable=true
        * Useful debugging properties:
            · mail.debug.auth=true
            · mail.debug.auth.username=true
            · mail.debug.auth.password=true

- Save the authorizer
- Obtain the initial consent
    - Click on *Get initial consent* button
    - Log into the email account
    - If a *testing* screen is presented, continue

  - Select/check Gmail access
  - Close the result screen with the refresh token and the access token

- The refresh and access tokens will appear in the disabled fields of the authorizer screen.

The proper configuration should look like the one in the figure 9.1.

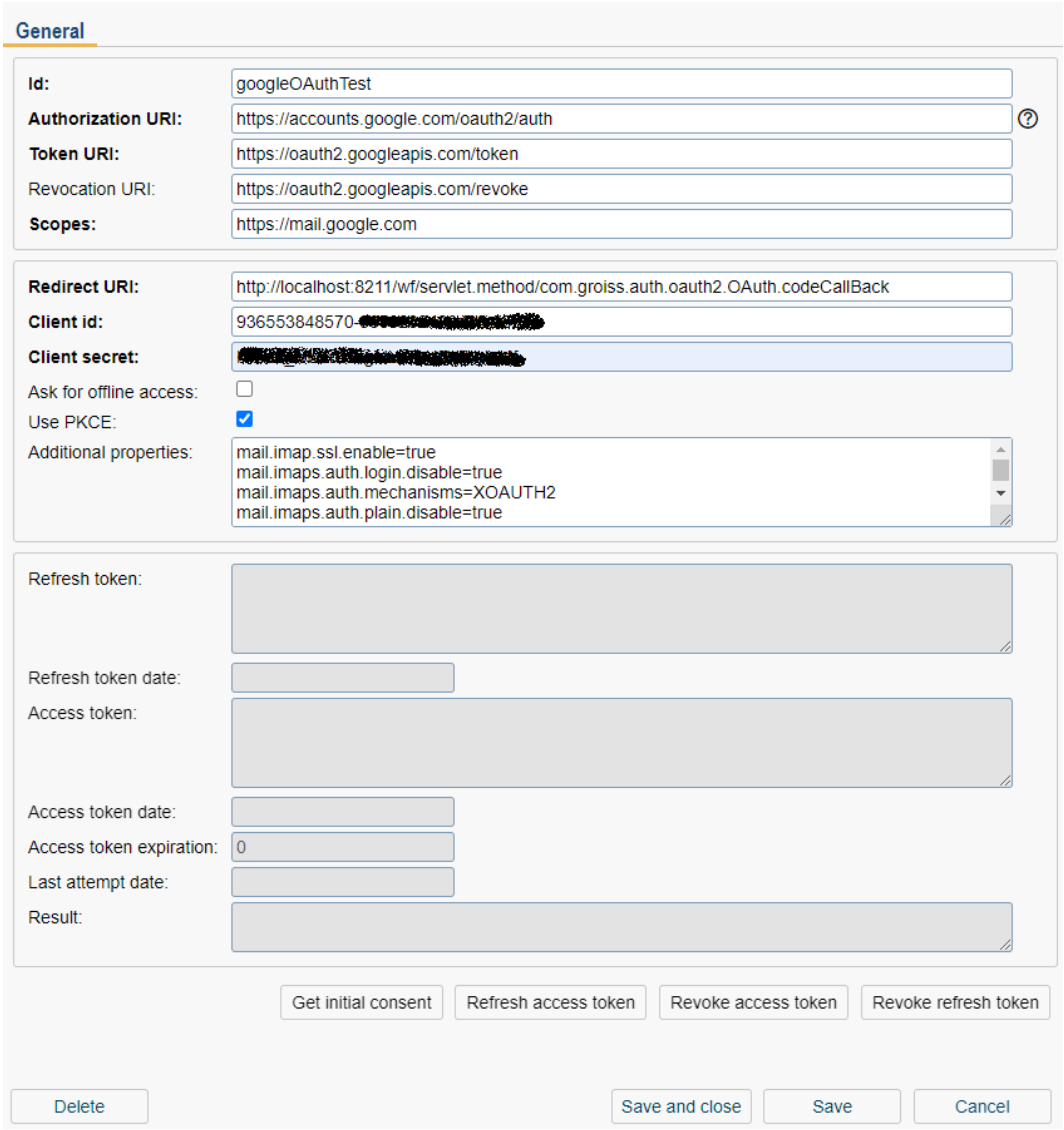

Figure 9.1: **Authorizer for Google**

9.2 *Specific Configuration for Microsoft Azure/Office365*

9.2.1 Client registration

- Log in to *https://portal.azure.com*

- Go to *Azure Active Directory*
  - Select *App-Registrations*
  - Select *New Registration*
  - Enter an appropriate name for the Client app
  - Choose *Accounts in this organizational directory only*
  - As the Redirect URI, select type *Web* and enter: *<protocol>://<host>:<port>/<context>/servlet.method/com.groiss.auth.oauth2.OAuth.codeCallBack*
    - Where *<protocol>*, *<host>*, *<port>* and *<context>* must be changed according to your *@enterprise* installation. This URL is not public, it must be a *locally resolvable* URI where the browser will be redirected to.
- Go to *Certificates and Secrets*
  - Select *New client secret*
  - Enter a description and validity period
    - Note down the secret value!
    - In particular, the value will never be displayed again!
- Go to *API-Permissions*
  - Select *Add a permission*
  - Choose *Microsoft Graph* and then *Delegated Permissions*
  - Add the following permissions:
    - OpenId-Permissions
      - openid
        - *https://graph.microsoft.com/openid*
      - Email
        - *https://graph.microsoft.com/email*
      - offline_access
        - *https://graph.microsoft.com/offline_access*
      - Profile
        - *https://graph.microsoft.com/profile*
    - IMAP
      - IMAP.AccessAsUser.All
        - *https://graph.microsoft.com/IMAP.AccessAsUser.All*
    - POP
      - POP.AccessAsUser.All
        - *https://graph.microsoft.com/POP.AccessAsUser.All*
    - SMTP
      - SMTP.Send
        - *https://graph.microsoft.com/SMTP.Send*
    - User
      - User.Read
        - *https://graph.microsoft.com/User.Read*

9.2.2 Authorizer configuration for Microsoft Azure/Office365

- In the *@enterprise* Admin Gui, navigate to *Admin tasks/Communication/Authorizers*

  NOTE: To automatically populate the fields in the authorizer use the question-mark icon near the *Authorization URI* field. Be sure to change the <*variables*> according to your specific installation.

- To manually create a new Authorizer please enter the following data:
  - Id:
    - An arbitrary id
  - Authorization URI:
    - *https://login.microsoftonline.com/<tenantid>/oauth2/v2.0/authorize*
      Be sure to change the <*tenantid*> with *Directory (tenant) ID* from the *Microsoft Azure* registered Application.
  - Token URI:
    - *https://login.microsoftonline.com/<tenantid>/oauth2/v2.0/token*
  - Revocation URI:
    - Not supported, leave it empty
  - Scope, enter a space separated list of:
    - *https://outlook.office365.com/IMAP.AccessAsUser.All*
    - *https://outlook.office365.com/POP.AccessAsUser.All*
    - *https://outlook.office365.com/SMTP.Send*
    - *offline_access*
  - Redirect URI:
    - *<protocol>://<host>:<port>/<context>/servlet.method/com.groiss.auth.oauth2.OAuth.codeCallBack*
      - Where <*protocol*>, <*host*>, <*port*> and <*context*> must be changed according to your *@enterprise* installation. This URL is not public, it must be a *locally resolvable* URI where the browser will be redirected to.
  - Client Id:
    - The client id obtained at client registration
  - Client secret:
    - The client secret obtained at client registration
  - Ask for offline access:
    - Do not check it
  - Use PKCE:
    - Check it
  - Additional properties: depends on intended usage and can be combined:

    - Properties for IMAP:
        - mail.imap.ssl.enable=true
        - mail.imaps.auth.login.disable=true
        - mail.imaps.auth.mechanisms=XOAUTH2
        - mail.imaps.auth.plain.disable=true
    - Properties for SMTP:
        - mail.smtp.starttls.enable=true
        - mail.smtp.auth.mechanisms=XOAUTH2
        - mail.smtp.auth.login.disable=true
        - mail.smtp.auth.plain.disable=true
    - Useful debugging properties:
        - mail.debug.auth=true
        - mail.debug.auth.username=true
        - mail.debug.auth.password=true

- Save the authorizer
- Obtain the initial consent
    - Click on *Get initial consent* button
    - Log into the email account
    - If a *testing* screen is presented, continue
    - Close the result screen with the refresh token and the access token
- The refresh and access tokens will appear in the disabled fields of the authorizer screen.

The proper configuration should look like the one in the figure 9.2.

9.3 Automatic token refresh

To periodically refresh the expiring access tokens, *@enterprise* provides the *TokenRefresh* timer. It must be activated: navigate to *Admin GUI / Applications / Default / Timer:*, select the *TokenRefresh* timer and activate it. Be sure to set an appropriate interval. It should be dependent on the token expiration interval. The timer will renew a token, if less than half of the expiration interval remains.
You can specify the Timeouts for the *TokenRefresh* timer. Properties in the timer can be used to separately specify connect timeouts and read timeouts (in seconds) like:

```
connectTimeoutDuration=20
readTimeoutDuration=30
```

If those properties are not present, a value of 0 will be used, which means potentially indefinite waiting. When a timeout on token refresh occurs, an error is logged and persisted in the result field of the authorizer object, no further action is taken.
In the authorizer mask, a new access token can be requested manually via the button *Refresh access tokens*. This might be handy for debugging purposes.

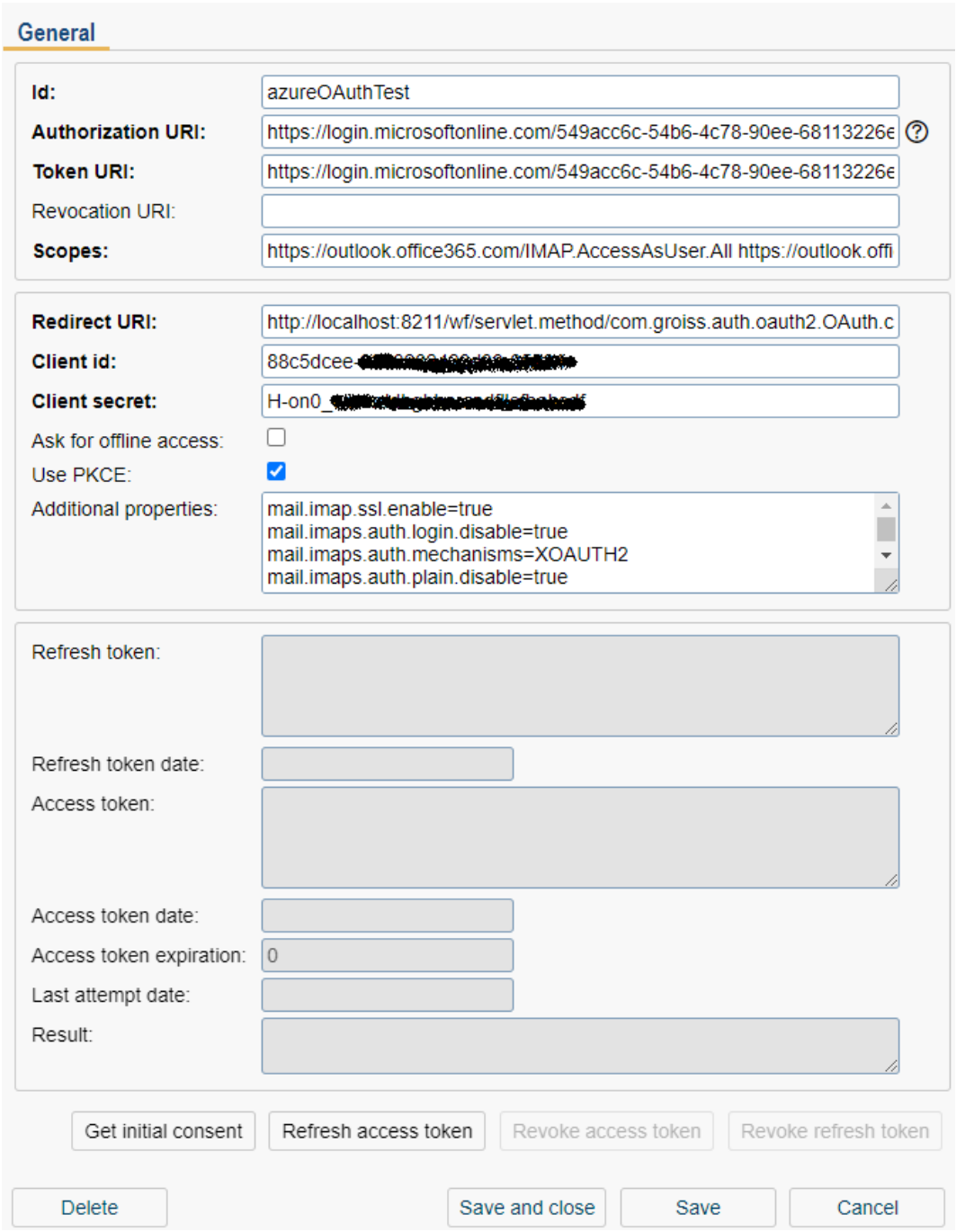

Figure 9.2: **Authorizer for MS Azure**

9.4 *Activating an authenticator for email reception*

After the authenticator has been configured and the initial consent has been granted, navigate to Mailboxes, select the mailbox, select the authenticator, and use an empty password. Save the change and try to view the contents of the mailbox by selecting the corresponding tab.

9.4.1 Configure Mailbox for Google/Gmail

To receive emails over OAuth email authentication in *@enterprise* you should create a Mailbox (see figure 9.3):

- **Id:** an id of the Mailbox.
- **Server:** *imap.gmail.com*
- **User:** Users Gmail address
- Password: Leave it empty!
- Authorizer: Chose the properly configured authorizer
- **Email address:** Users Gmail address
- Mail Protocol: IMAP
- Type of communication: Encrypted
- SMTP host: *smtp.gmail.com:465*
- Type of SMTP communication: Encrypted

More information about how to create a mailbox in *@enterprise* can be found in *System Administration Guide* in section *Administration tasks/Communication/Mailboxes*.

9.4.2 Configure Mailbox for Microsoft Azure/Office365

To receive emails over OAuth email authentication in *@enterprise* you should create a Mailbox (see figure 9.4):

- **Id:** an id of the Mailbox
- **Server:** *outlook.office365.com*
- **User:** Users email address
- Password: Leave it empty!
- Authorizer: Chose the properly configured authorizer
- **Email address:** Users email address
- Mail Protocol: IMAP
- Type of communication: Encrypted
- SMTP host: *smtp.office365.com:587*
- Type of SMTP communication: STARTTLS

More information about how to create a mailbox in *@enterprise* can be found in *System Administration Guide* in section *Administration tasks/Communication/Mailboxes*.

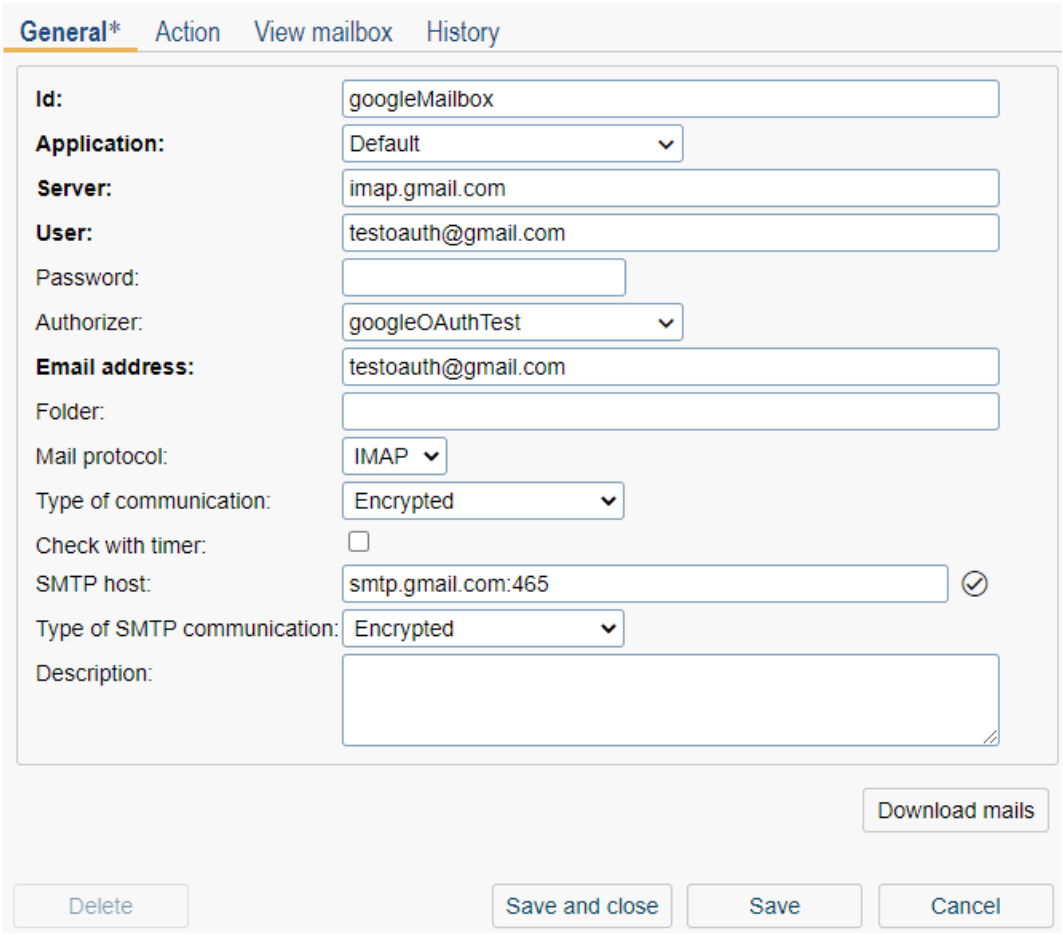

Figure 9.3: **Mailbox for Google**

9.5 *Activating an authorizer for sending mails*

At *Administation/Configuration/Communication*, enter the id of the authorizer, remove the SMTP password. Save the screen and send a test mail via the tick icon to the right of the Administrator email address.

9.5.1 Communication configuration for Google/Gmail

To send emails over OAuth email authentication in *@enterprise* some configurations in *Administration/Configuration/Communication* have to be carried out (see figure 9.5):

- SMTP host: *smtp.gmail.com* (there may be the need to set a port to 465 - *smtp.gmail.com:465*)
- Mail sender: Client Gmail address
- SMTP Username: Client Gmail address
- SMTP Password: Leave it empty!

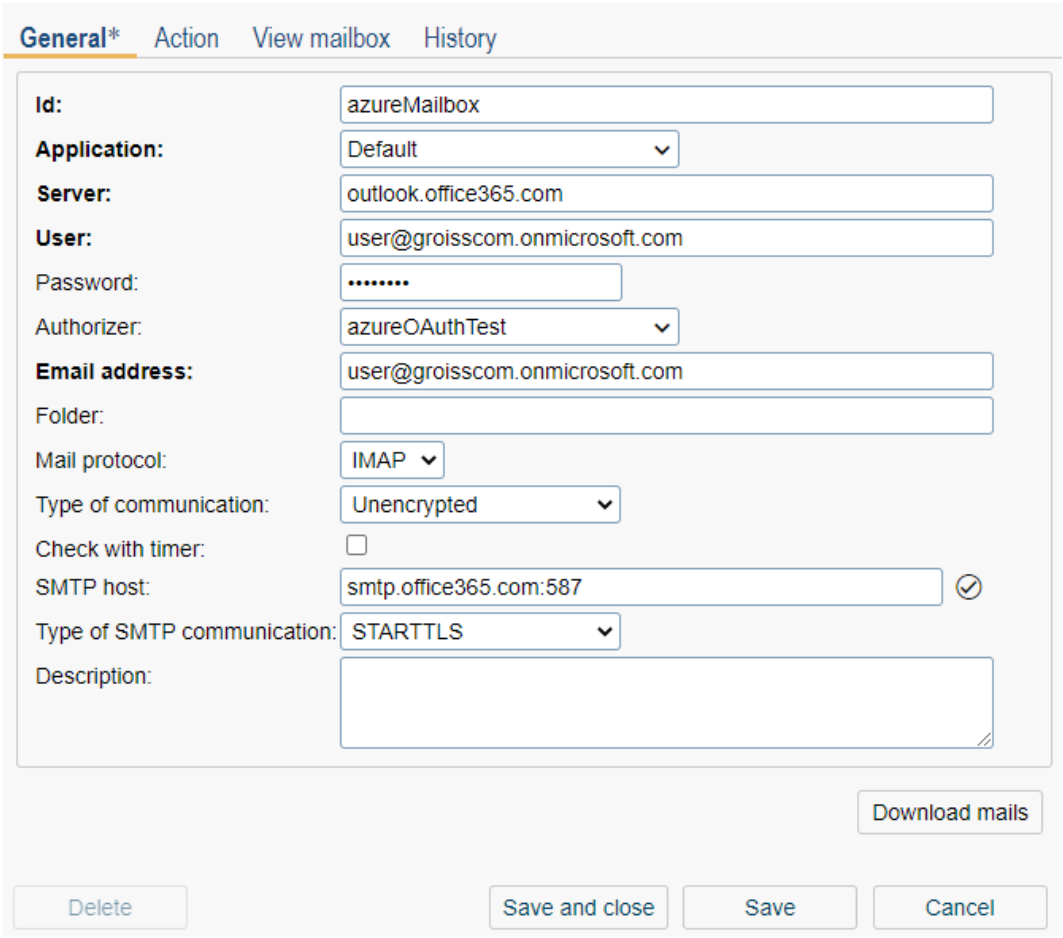

Figure 9.4: **Mailbox for MS Azure**

- SMTP Authorizer: the ID of the properly configured authorizer
- Type of SMTP communication: Encrypted

More information about communication properties can be found in section Communication.

9.5.2 Communication configuration for Microsoft Azure/Office365

To send emails over OAuth email authentication in *@enterprise* some configuration in *Administration/Configuration/Communication* have to be carried out (see figure 9.6):

- SMTP host: *smtp.office365.com* (there may be the need to set a port to 587 - *smtp.office365.com:587*)
- Mail sender: User's registered email address
- SMTP Username: Users registered email address
- SMTP Password: Leave it empty!
- SMTP Authorizer: the ID of the properly configured authorizer

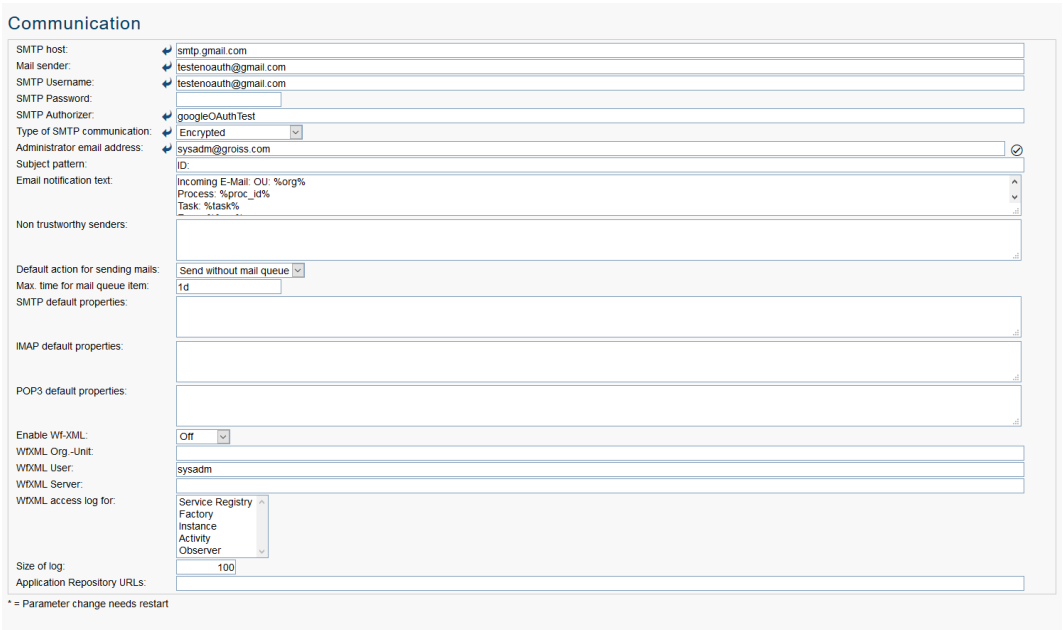

Figure 9.5: **SMTP configuration for Google**

- Type of SMTP communication: STARTTLS

Hint: If there is an error about *"SmtpClientAuthentication is disabled for the Tenant"* during an attempt to send an email, then please visit *https://aka.ms/smtp_auth_disabled* and act accordingly to the recommendations found there.
More information about communication properties can be found in section Communication.

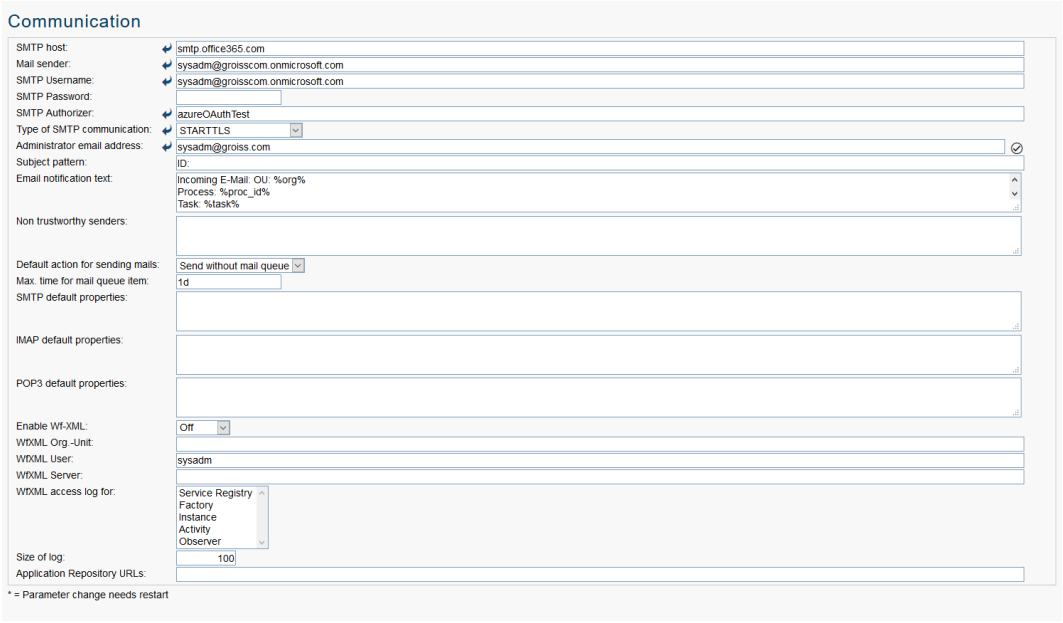

Figure 9.6: **SMTP configuration for MS Azure**

A Hints for Server Sizing

A.1 General remarks for Server sizing

The following section provides a basis for a rough but reasonable estimation of key operatiing requirements for deploying @*enterprise* in typical use cases. Please be aware that the real requirements can significantly differ depending on a lot of factors which are specific to the deployment.
Adaption of the results for productive environments will definitely be required and should based on the actual emerging usage patterns. For the estimations, we assume that the DB-server and application-server are running on dedicated machines. If they are running on the same machine, the respective numbers need to be added up.
The numbers are just for the purposes of @*enterprise* itself. Resource requirements for the underlying platforms themselves have to be added (i.e. for Java application servers and the operating system itself).

A.2 Application Machine

A.2.1 Disk space

For the disk space, we can give the following estimation:

- basic space of 300 MB
- plus additional disk space for JDK or third-party application-server
- plus space for database client (esp. if Oracle is being used together with the OCI-driver)
- plus space for log-files, depending on log-level, 10 MB to several GB per day (may periodically be deleted)
- plus space for temporary files: 100 MB + maximum size of document expected to be loaded * number of threads simultaneously uploading

An alternative estimation for disk space is:

- 5GB
- plus 100 MB * (number of concurrent users)

At least for the log files, low-latency devices (SSDs) are recommended.

A.2.2 Processor

For the processor, we give the following rough estimation:

- at least 1 Core with 500 MHz
- plus 50 MHz * (number of concurrent users)

A.2.3 Main memory

The memory size can be estimated via:

- 1 GB
- plus 2 MB * (total number of registered users)
- plus 4 kB * (number of active processes)
- plus (maximum size of a document) * (number of threads simultaneously editing documents)

A.2.4 Network connection

The connection bandwidth should be at least 1GBit; depending on actual network load, a dedicated connection between application server and DB server may be required.

A.3 Database Machine

The requirements do depend very much upon the concrete DBMS-product, nevertheless, the following estimations are a good starting point:

A.3.1 Disk space

For the disk space, we can give the following estimation:
(number of process-steps) * (number of process-instances) * (20 kB + size of process forms) + overall size of attached documents

A.3.2 Processor

Same as for application machine.

A.3.3 Main memory

The memory size depends on the number of DB-connections. A reasonable recommendation for the number of DB-Connections is : 10 + (number of concurrent users)/3.
For main memory the estimation is:

- 1GB
- plus 1 percent of disk space
- plus 10 MB * (number of DB-connections)

A.3.4 Network connection

The connection bandwidth should be at least 1GBit; depending on actual network load, a dedicated connection between application server and DB server may be required.

A.4 Example

The following table gives examples for the estimations for 50, 100 and 250 concurrent users respectively. It is based on the following assumptions:

- CPU:
  processor cores run at 3.2 GHz
- main memory for application server:
  1 GB+2 MB*Named Users+4 KB* Named Users*25 Active WorkItems/User
- network:
  1 Adapter for HTTP-Traffic, 1 Adapter for DB Connections
- disk space of database machine:
  documents + form data + workflow data for 1 year:
  - 2500 documents of 20 MB each
  - start of 5 processes per user and day
  - 20 steps per process
  - 20 KB of workflow data per step
  - 4 KB of form data per step
  - 200 working days per year

	Concurrent Users	50	100	250
	Named Users	100	200	500
Application Machine	Disk (GB)	10	15	30
	CPU (Cores*GHz)	3	5.5	13
	Number of cores	1	2	4
	Main memory (MB)	1210	1525	2050
	Network Adapters (GB Ethernet)	2	2	2
Database Machine	Disk (GB)	98	170	290
	CPU (Cores*GHz)	3	5,5	13
	Number of cores	1	2	4
	DB-Connections	30	50	100
	Main memory (MB)	2250	3150	4850
	Network Adapters (GB Ethernet)	1	1	1

B Database Performance Hints under Oracle

B.1 Preliminaries

The statements in this chapter refer to an *@enterprise* installation with an Version 8 Oracle DBMS. It is assumed that no atypical characteristics concerning either data distributions or data volumes or transaction volumes like extremely long worklists or BLOBs dominate the system. Further we assume that no other significant workload besides the *@enterprise*-service is processed on the system (dedicated hardware).

For successful performance improvements, the most crucial issue is to correctly identify and pinpoint system bottlenecks. Applying tuning actions without having a specific hint about the kind or reason for unacceptable performance is not target-oriented. It is essential to isolate and contain the problem area (database, *@enterprise* server, CPU, memory, network, own application classes, specific user operations). One should apply all means and tools which are offered by the underlying platform to check performance parameters or monitor them on a regular basis. Because of the wide variety of the platforms concerning this specific area, we refer the reader to the appropriate systems documentation.

We assume that the reader has some basic familiarity about the architecture of Oracle and is somewhat acquainted with its significant mechanisms.

B.2 Key Operating Parameters of the Database

The following parameters are vitally important for an efficient operation of the database. They all can be found in the **ini.ora** file.

DB_BLOCK_SIZE States the size of the data blocks in the DB. In most environments the default value is 2048 bytes. For *@enterprise* the value should be increased to 4096 or 8192. The change should reduce IO-overhead and has no other significant implications. Unfortunately, the value can't be changed in an existing data base, one would be forced to apply a complete export/import cycle to apply a modification.

DB_FILE_MULTIBLOCK_READ_COUNT Determines how many blocks are read during a full table scan. The value should be dimensioned in such a way, that the product of DB_BLOCK_SIZE and DB_FILE_MULTIBLOCK_READ_COUNT equals the size of the operating system buffer (often 64K). The value can be changed during operations but is applied only at the next startup of the database instance.

DB_BLOCK_BUFFERS States the size of the database block buffer caches in units of blocks. It is an extremely crucial parameter. The default values of Oracle are way to small. For an application system with the characteristics of *@enterprise* (mostly interactive users in OLTP, insignificant batch processing) one should configure the cache size to achieve a hit rate above 95% to 98% in regular operations. Regular monitoring is essential. One could apply the following queries (as user SYSTEM) to determine current hit rates:

```
select
   SUM(DECODE(Name, 'consistent gets',Value,0)) Consistent,
   SUM(DECODE(Name, 'db block gets',Value,0)) Dbblockgets,
   SUM(DECODE(Name, 'physical reads',Value,0)) Physrds,
   ROUND(((SUM(DECODE(Name, 'consistent gets', Value, 0))+
      SUM(DECODE(Name, 'db block gets', Value, 0)) -
      SUM(DECODE(Name, 'physical reads', Value, 0)) )/
       (SUM(DECODE(Name, 'consistent gets',Value,0))+
        SUM(DECODE(Name, 'db block gets', Value, 0))))
        *100,2) Hitratio
from V$SYSSTAT;
```

```
column HitRatio format 999.99
select Username,
       Consistent_Gets,
       Block_Gets,
       Physical_Reads,
       100*(Consistent_Gets+Block_Gets-Physical_Reads)/
           (Consistent_Gets+Block_Gets) HitRatio
  from V$SESSION, V$SESS_IO
 where V$SESSION.SID = V$SESS_IO.SID
   and (Consistent_Gets+Block_Gets)>0
   and Username is not null;
```

If an unsatisfactory hit rate is measured, DB_BLOCK_BUFFERS should be increased in steps of 15% to 25%, until hit rate levels out. Meaningful measurements are only possible in real production mode and not immediately after the startup phase of the instance when the cache is still cold.
It is common knowledge, that the buffer cache should not be increased beyond certain thresholds. Each word of main memory that is allocated exclusively for the buffer cache can be in high demand by other system components. In no way the machine should be pressed to swapping or paging activities. After every expansion of buffer cache size, measurements with a warm cache are called for in combination with keeping an eye on paging or thrashing. Memory expansions should be considered at such points.

SHARED_POOL_SIZE Determines the size of the shared pool in the System Global Area (SGA). Oracle defaults are often found to be too small.
A rule of thumb says that 15% to 20% of the shared pool should stay free.
The current size can be calculated as follows:

```
select value from v$parameter where name='shared_pool_size';
```

The free space is returned by this query:

```
select name, bytes from v$sgastat where name='free memory';
```

Key elements in the shared pool are the library cache and the data dictionary. Miss rates for both components can be determined with the help of the following queries. In the library cache miss rates of under 1% and of under 5% in the data dictionary are commonly seen as appropriate.

```
column "Executions" format 9,999,999,990
column "Cache Misses Executing" format 9,999,999,990
column "Data Dictionary Gets" format 9,999,999,999
column "Get Misses" format 9,999,999,999
column "% Ratio" format 999.99



select sum(pins) "Executions",
       sum(reloads) "Cache Misses Executing",
       (sum(reloads)/sum(pins)*100) "% Ratio"
from v$librarycache;


select sum(gets) "Data Dictionary Gets",
       sum(getmisses) "Get Misses",
       100*(sum(getmisses)/sum(gets)) "% Ratio"
from v$rowcache;
```

If higher mis rates are measured, we advise a similar procedure like in the case of the DB_BLOCK_BUFFERS parameter.

SORT_AREA_SIZE Size of the area in the main memory which is reserved for each user for in-memory sorting operations. If disk-based sorts make up for more than 5% to 10% of the in memory sorts, then SORT_AREA_SIZE should be increased. The current configuration can be determined with:

```
select substr(name,1,25) Name,
        substr(value,1,15) Value
  from V$PARAMETER
 where Name = 'sort_area_size';
```

Statistics about the number of sorts, separately for main memory and disk based sorts are implemented by:

```
select substr(name,1,25) Name,
        substr(value,1,15) Value
  from V$SYSSTAT where name like 'sort%';
```

LOG_BUFFER Size of the redo log buffer in the SGA.
The current size can be obtained by:

```
select substr(name,1,25) Name,
        substr(value,1,15) Value
  from V$SGA
 where Name = 'Redo Buffers';
```

If redo log space requests are issued in the database, there might be a bottleneck here. The following query investigates this:

```
select  substr(name,1,25) Name,
        substr(value,1,15) Value
from v$sysstat
where name = 'redo log space requests';
```

The value should approximate zero. If this is not the case, one should increase the LOG_BUFFER parameter in steps of 50% to 100%. It might be advisable to increase the shared pool size by the same (absolute) amount.

B.3 Optimizer

Cost based optimization is the way to go with Oracle. In general, better query plans can be generated than pure rule based optimization could achieve.
To activate the cost based optimizer, the parameter OPTIMIZER_MODE in init.ora must be set to CHOOSE. It is also necessary to statistically analyze the data distribution and index selectivity.
Oracle offers commands of the form `analyze table <mytable> compute statistics`. One can supplement statistics for an entire schema using
`execute dbms_utility.analyze_schema('USER','COMPUTE');`. The 'USER' element should be replaced by the name of the *@enterprise* data base user.
It is highly advisable to run this command from time to time. In any case, it should be run periodically during the first period of production use and additionally when significant configuration changes (new applications, other data volumes) take place. The analysis is quite resource intensive and should not be applied during peak operational hours. Sufficient temporary tablespace must be provided, also. A practical trade-off between statistical accuracy and resource consumption can be achieved through use of 'ESTIMATE' instead of 'COMPUTE'. In this case the system takes samples of the data and does not go through the entire volume. A good strategy might be to establish a batch-job which issues this schema analysis commands on a regular (weekly) basis.

B.4 Storage

B.4.1 Disks

The main performance issues in the disk subsystem are the separation of random access and sequential access and further to isolate individual sequential accesses.
More precisely, separate the redo-logs, the after image files and the rollback segments, and put them on individual disks without any further activity.
Further split up SYSTEM and TEMPORARY tablespaces from the rest of the system.
Tables with particular high activity on them are AVW_STEPINSTANCE, AVW_FOLLOWS and AVW_FORMVERSION. A good measure would be to place them together with their indices on separate tablespaces, to be able to place them on specific disks and to distribute the load on multiple devices. Another possible strategy would be the division of index space and table data space in different tablespaces.
It is not possible to give general advice without deeper knowledge of the operational characteristics. Nevertheless, for an installation with significant size, we strongly recommend to devote some thoughts to this issues and to divert from the default configuration.
An overview about IO distribution over the individual data files can be gained by:

```
select DF.Name File_Name,
       FS.Phyblkrd Blocks_Read,
       FS.Phyblkwrt Blocks_Written,
       FS.Phyblkrd+FS.Phyblkwrt Total_IOs
  from V$FILESTAT FS, V$DATAFILE DF
 where DF.File#=FS.File#
 order by FS.Phyblkrd+FS.Phyblkwrt desc;
```

B.4.2 Parameters for Tablespaces

Appropriate default storage parameters for the tablespaces would be:

```
alter tablespace AVW default storage
   (initial 256k next 256k maxextents 200 pctincrease 0);
```

Instead of AVW, state the tablespaces which are used to store the *@enterprise* tables and indexes, in particular the default tablespace of the *@enterprise* database user. For some tables which can be assumed to have a greater size than that (50 MB) like AVW_STEPINSTANCE, AVW_FOLLOWS and AVW_FORMVERSION, the storage parameters can be changed in full operation mode; e.g.:

```
alter table <mytable> storage(next 1M maxextents 1200);
```

With this statement, table <mytable> can use 1000 additional extents, each being 1 MB in size when one assumes that 200 extents were already used. It is generally advisable to use zero as value for `pctincrease`, to avoid exponentially increasing storage demand for extents.

B.5 One owns Tables and Queries

For own tables which are used to store application relevant data, exactly the same considerations like for system tables according to table placement and to storage parameters should be made. In particular, popular access paths should be supported by appropriate (multi-column) indexes.
Queries of application tables should generate a result set as small as possible. It is recommended to use a two phase approach for queries with potentially large result sets. First, the number of tuples (count(*)) should be determined. If this number exceeds a certain threshold, it is time to give the user a chance to decide upon further execution of the query. The user could apply additional constraints to the search condition which would further confine the result set, or she could explicitly get the whole large result set (and thereby accepting higher response time and workload on the server).
For medium sized tables, which are often scanned in their entirety, table level caching could be advantageous:

```
alter table mytable cache;
```

Clearly, sufficient space in form of DB_BLOCK_BUFFERS must be provided.
Criteria in queries should be used in such a way that indexes get used. Strive for point queries or at least for multipoint queries with high selectivity. (its better to use `a='b'` than `a like 'b%'` which is in turn better than `a like '%b%'`).
Of uttermost importance is the usage of the *@enterprise* transaction cache mechanism, which works for all subclasses of `SQLObject`. Access to such objects should be done through `receiver.get(oid)` and not via `receiver.get('oid=xxx');`.
Performance friendly formulation of application queries (especially such statements which are executed quite often) call for generation, interpretation and perhaps modification of the execution plans. Measures could be the definition of additional indices or clustering on a physical level or semantic preserving reformulation of the query or explicit incorporation of query optimization hints.
Concerning these issues we refer to the 'Oracle8 Tuning' and 'Oracle8 Concepts' and 'Oracle8 Application Developers Guide' manuals. Consider the possibilities of TKPROF and EXPLAIN PLAN. The log file of *@enterprise* may have valuable first hints like duration of SQL statements.
It is much better to run complex queries in their entirety on the DB-server than to overflow the server with lots and lots of simple individual queries and to stick their results together in the *@enterprise* server. This is due to relatively high startup and communication overhead and context switches between the two servers.