Administration Guide on Platypus.js

Administration Guide on Platypus.js

Table of Contents

List of Tables

Chapter 1. Platform.js’s installation

If the Platypus.js runtime is delivered as a ZIP file, unpack its contents to a location according to your preference on your hard drive.

> **Note**
>
> Instructions for installing the Platypus Application Designer developer tool are available in the Development Guide.

1.1. System requirements

Below you can see prerequisites for launching the platform components on your computer:

- 32-bit (x86) or 64-bit (x64) processor running at 1 gigahertz (GHz) or higher
- 1 gigabyte (GB) (for 32-bit system) or 2 GB (for 64-bit system) RAM
- 5 gigabytes (GB) of free hard disk space
- Windows, Linux, Mac OS X or other operating system like Unix
- JDK 8+
- Graphical user interface for client operation
- Internet access for downloading updates and working with external mapping service (In a corporate network, Internet access may be arranged through a proxy server.)
- Google Chrome 19, Mozilla Firefox 10, Internet Explorer 9 or newer versions of browsers for Platypus HTML5 client operation

The Platypus Platform supports the following database servers:

- Oracle Database 10g and higher
- IBM DB2 9 and higher
- Microsoft SQL Server 2008 and higher
- MySQL (InnoDB) 5.5 and higher
- PostgreSQL 9 and higher
- H2 Database 1.3 and higher

For information on installing and configuring the database server, refer to the relevant installation and administration manuals, provided by database server manufacturers.

The platform runtime is supplied with the H2 database. This database does not require any additional installation and configuration steps.

For running applicatons on server side in a J2EE servlet container use a J2EE server or container, for example:

- Apache Tomcat 8.0.24 or higher
- WildFly 9.0.0 or higher

For information on installing and configuring the servlet container or J2EE server, refer to the relevant installation and administration manuals, provided by their manufacturers.

1.2. Installation guide

To install the Platform.js runtime, unpack *.zip file you have downloaded to the desired directory.

After installation, the platform’s directory will have the following structure:

- `platypus/`
  - `api/`
  - `bin/`
  - `etc/`
  - `ext/`
  - `lib/`
  - `updates/`

You might whant to install Platypus.js runtime as part of All-in-one bundle installer. If so, perform the following actions:

1. Run downloaded installer file.
2. Select components (Platypus.js runtime, Platypus Application Designer, Apache Tomcat server) to be installed.
3. Select the installation directories.

1.3. Uninstallation guide

If you have unpacked platypus.js from *.zip file manually, delete platypus directory from your hard disk. If you have installed platypus.js using All-in-one bundle installer, than perform the following actions:

- Select the Platypus.js runtime menu item from the installed programs menu. For Windows, use the conventional mechanism for removing programs.
- Confirm uninstalling of Platypus.js runtime. If necessary, enable an option for removing the working directory.

Chapter 2. Application management

2.1. Security

Platypus platform is equipped with security mechanisms and provides restricted access to system resources based on roles. Roles are introduced on application level.

As for authentication, Platypus may use various security domains. The security domain may be internal or external and contain information about users and their group membership. The following security domain options are avaliable:

- An database users registry, which is located in two database tables. This option provides simple tools for storing information about users and groups.
- External storages of authentication data, for example, a LDAP server (Active Directory, OpenLDAP, etc.).

The security domain is used to define a set of groups or global roles for the user, which can be associated with roles at the application level.

When using database users registry mode, user information is stored in the `MTD_USERS` table of the application database. Information about the groups, which the user belongs to, are stored in the `MTD_GROUPS` table.

The `MTD_USERS` table contains the following mandatory fields:

Field	Description
`USR_NAME`	Username
`USR_PASSWD`	Hash sum of the user password using the MD5 algorithm

In addition, the `MTD_USERS` table can include optional fields containing additional information about the user.

The `MTD_GROUPS` table contains the following mandatory fields:

Field	Description
`USR_NAME`	Username
`GROUP_NAME`	Group name

To init the users and user groups tables use the management utility command line with the the following options:

Table 2.1. Command for creating users and users group table

Parameter	Description
`-initusers`	Checks and initializes users database store tables if they are not initialized

After initialization users and groups tables are filled with the default credentials as follows: `admin` as username and `masterkey` as a password. The `admin` user is a member of the `admin` group.

> **Important**
>
> Change the default username and password before shipping your application for production.

Platypus applications deployment

```
Deployment is the process of spreading the finished application for
installing or upgrading on the production server. On the production
server, the Platypus application resides on the hard drive in the form
of a directory.

Platypus applications are provided by the following ways:

* On Flash, CD or DVD drive.
* Over the Internet as a zip archive file or as war archive Java EE Web application.

If the Platypus application is delivered as a zip file, unpack its
contents to a location according to your preference on your hard drive.

After unpacking, the directory will have the following structure:

* `appDirectory/`
** `META-INF`/
** `WEB-INF`/
** `app/`
** `pub/`
** `web/`
** `application-start.html`
** `login-failed.html`
** `login.html`
** `private.properties`
** `project.properties`

The Platypus application directory corresponds to the application
project, which is ready for deployment and execution, as well as for
modification in Platypus Application Designer.

The application directory includes the `project.properties` configuration
file, the `app` directory, containing application elements, `META-INF` and `WEB-INF`
directories, containing servlet container configuration for the application and
Java EE web.xml configuration respectively and `private.properties` designer configurati

Platypus.js application projects directories structure is ready for use as *.war archive
You may simple make a zip archieve and rename it to *.war and deploy it to Java EE or Se
in a standard way.

NOTE: If application files are named using cyrillic letters, than `jar` utility must be

[[realm-built-in]]
Built in realm
```

When using Platypus.js TSA Server it is possible to use built-in users space, which is represented as two database tables: `MTD_USERS` and `MTD_GROUPS`. `MTD_USERS` is table of users and `MTD_GROUPS` is table of groups/roles. For additional information, see the "Security" chapter. To initialize theese tables

in a database Deploy tool might be used. When using Java EE or Servlet container, it is recommended to use standard Jdbc, File or LDAP RelaMs supplied by container.

The command for initializing the in database users space is as follows:

```
java -jar Deploy.jar OPTIONS
```

where OPTIONS is a set of its parameters.

Table 2.2. Init users space command parameters

Parameter	Description
`-initusers`	Task marker command. There are some other experimental tasks. So `-initusers` is needed to be used explicitly.
`-url`	Jdbc url to target database.
`-dbuser DB_USER`	Username for authorization in the database.
`-dbpassword DB_PASSWORD`	User password for authorization in the database.
`-dbschema DB_SCHEMA`	Database scheme.

Chapter 3. Java SE client

The JavaSE client is a Java SE swing application to provide user interface and/or execute the application logic on the end user's computer. The JavaSE client loads required application elements from the application server, or directly from a disk, according to its configuration.

The JavaSE client supports automatic updates via network.

3.1. Running Java SE client

Startup command for Platypus client typically is as follows: java -D.level=INFO -Dhandlers=java.util.logging.ConsoleHandler -Djava.util.logging.ConsoleHandler.level=INFO -Djava.util.logging.ConsoleHandler.formatter=com.eas.util.logging.PlatypusFormatter -Djava.util.logging.config.class=com.eas.util.logging.LoggersConfig -cp "C:\Program Files\PlatypusJs\bin\Application.jar;C:\Program Files\PlatypusJs\api;C:\Program Files\PlatypusJs\ext/*;C:\Program Files\PlatypusJs\ext" com.eas.client.application.PlatypusClientApplication -url http://localhost:8080/proba

If command-line argument `url` is missing, dialog box for entering the user name and password and selecting a preconfigured connection to the server is displayed.

If the no-server configuration is used, enter the username and password of the database connection, and the username and password.

If the configuration with a server is used, enter only the username and password.

Use the Remember database password and Remember password check boxes, if it is necessary to remember the database password and/or Platypus applications password.

Select a preconfigured connection from the list to connect to the server and run the client.

To display the dialog for creating a connection, click the New button on the preconfigured connections panel.

To change a preconfigured connection, select it from the list and click the Change button on the preconfigured connections panel.

Enter or modify the following fields in the Connection settings dialog:

- Name is the connection name, for example `myserver`; you can enter any name in this field.
- Connection URL is the string of connection to the server. For working on the 2-tier scheme, the connection URL should be in the JDBC URL format, for example, `jdbc:oracle:thin:@dbhost:1521:adb`.

  For working on the 3-tier configuration over the Platypus Protocol, use URL of the `platypus://` format, for example, `platypus://serverhost:8500`, port 8500 is the default port for the Platypus protocol and in this case it can be omitted.

  For working on the 3-tier confirguration over the HTTP protocol, use URL of the `http://` format, for example, `http://localhost/myapp/application`. * Scheme is the default database scheme,

for example, `myschema` (not used in case of the 3-tier configuration). * Database user is the default database username, for example, `user1` (not used in case of the 3-tier configuration). * To save the new connection settings, click OK in the dialog, to cancel, click Cancel.

To remove a preconfigured connection, select the connection from the list and click the Delete button on the preconfigured connections panel. The connection is deleted when the user clicks OK in the removal confirmation dialog.

To run the client using the selected connection, click OK. To close the connection selection dialog, click Cancel.

3.2. Command line options

To configure starting of the desktop client, edit the contents of launch shell files.

The deskop client is a Java SE Swing application. To customize it, specify the startup JVM options and the applications parameters.

The command for running the Platypus client is as follows:

```
java JVM_OPTIONS -cp Application.jar;EXT_CLASSPATH com.eas.client.application.PlatypusCl
```

where JVM_OPTIONS is Java Virtual Machine options, EXT_CLASSPATH are paths which should be added to the Java class loader search path, OPTIONS are additional running parameters.

To specify the look and feel (L&F) in command line use the -D flag to set the swing.defaultlaf property, for example, to acitvate the Nimbus L&F add the following in the JVM_OPTIONS: `-Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel`.

For the information how to specify the application's current log level refer to Java documentation.

Table 3.1. Command line parameters

Parameter	Description
`-url URL`	URL to the applicaion. Can be one of the follow: `file://` to specify to a directory or `jndi://` to specify to a datasource.
`-appElement APP_ELEMENT`	Used only in two-tier configuration. Module or form element's name to run.
`-user USER_NAME`	Username for logging to the application.
`-password PASSWORD`	User password for logging to the application.
`-default-datasource DEFAULT_DS`	Name of the application's default datasource.
`-datasource DS`	Datasource name.
`-dburl DB_URL`	JDBC URL database connection.
`-dbuser DB_USER`	Username for authorization in the database.
`-dbpassword DB_PASSWORD`	Password for authorization in the database.
`-dbschema DB_SCHEMA`	Database scheme (optional).

Define zero or more datasources for a single application. A datasource is represented in a group of the following parameters: `-datasource`, `-dburl`, `-dbuser`, `-dbpassword` and optional `-dbschema` provided jointly. One of the datasource can be specified as a default using the `-default-datasource` parameter.

If database connection parameters or some credential are not specified, the Connection settings dialog appears.

To specify the look and feel (L&F) in command line use the -D flag to set the swing.defaultlaf property, for example: -Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel

The example of J2SE client running command in two-tier configuration:

```
java -D.level=INFO -Dhandlers=java.util.logging.ConsoleHandler -Djava.util.logging.Conso
```

The example of J2SE client running command in three-tier configuration with web server:

```
java -D.level=INFO -Dhandlers=java.util.logging.ConsoleHandler -Djava.util.logging.Conso
```

The example of J2SE client running command in three-tier configuration with Platypus TSA server:

```
java -D.level=INFO -Dhandlers=java.util.logging.ConsoleHandler -Djava.util.logging.Conso
```

The -Djava.util.logging.config.class=com.eas.util.logging.LoggersConfig argument acts as follows: Java system property `java.util.logging.config.class` assigned the following value: com.eas.util.logging.LoggersConfig. This allows to specify loggers configuration directly in command line instead of editing of logging.properties file. The following arguments: * -D.level=INFO * -Dhandlers=java.util.logging.ConsoleHandler * -Djava.util.logging.ConsoleHandler.level=INFO * -Djava.util.logging.ConsoleHandler.formatter=com.eas.util.logging.PlatypusFormatter are the same properties as in logging.properties file.

Chapter 4. Platypus Application Server

The main purpose of use of the Platypus Application TSA Server is to provide support for various binary communication protocols.

4.1. Running server

To start the server application, you run the startup shell script for the server, which contains java command and its command line arguments.

4.2. Command line options

The command for running the server is as follows:

```
java JVM_OPTIONS -cp EXT_CLASSPATH com.eas.server.ServerMain OPTIONS
```

where JVM_OPTIONS is Java Virtual Machine options, EXT_CLASSPATH are paths to be added to the Java class loader search path, OPTIONS are server running parameters.

Table 4.1. Command line options

Parameter	Description
`-iface`ADRESS:PORT,ADRESS:PORT	Network interfaces and listening ports for the server.
`-url URL`	URL to the applicaion. Can be one of the follow: `file://` to specify to a directory or `jndi://` to specify to a datasource.
`-appelement APP_ELEMENT`	Module or form element's name to run.
`-protocols` PORT:PROTOCOL,PORT:PROTOCOL	Network protocols for listening ports.
`-default-datasource DEFAULT_DS`	Name of the application's default datasource.
`-datasource DS`	Datasource name.
`-dburl DB_URL`	JDBC URL database connection.
`-dbuser DB_USER`	Username for authorization in the database.
`-dbpassword DB_PASSWORD`	Password for authorization in the database.
`-dbschema DB_SCHEMA`	Database scheme (optional).

Define zero or more datasources for a single application. A datasource is represented in a group of the following parameters: `-datasource`, `-dburl`, `-dbuser`, `-dbpassword` and optional `-dbschema` provided jointly. One of the datasource can be specified as a default using the `-default-datasource` parameter.

The example of the application server running command:

```
java -D.level=INFO -Dhandlers=java.util.logging.ConsoleHandler -Djava.util.logging.Conso
```

Chapter 5. Java EE server

The plaftorm’s server components can run in a J2EE server or a servlet container.

This configuration has the following features:

- Server components are deployed in the J2EE container/on the server as a web-application as a WAR-archive or a folder. Special sevlet provides interaction with the clients.
- Desktop client and HTML5 browser are supported as clients.
- It is possible to use an external user authentication service, such as Microsoft Active Directory service. It allows you to integrate the platform target application in existing enterprise users space.
- Database connecions are configrued as a JNDI resources.
- Application sever modules methods are avaliable via HTTP.

  Note

  Use Platypus Application Designer to create web application. For detailed information, refer to the Development Guide.

5.1. Configuring J2EE server

To run the plaform’s application in a J2EE servlet container or on the application server, perform the following actions:

- Create a new directory with a standard structure for a web-application in J2EE, including the `WEB-INF/web.xml` deployment descriptor.
- Create HTML pages, which will contain the Platypus application. Configure JavaScript code, necessary for the initial startup code.
- Copy libraries, necessary for the application functioning, to the web-application `WEB-INF/lib` directory from the `bin` and `lib` sub-directories of the Platypus.js runtime directory.
- Copy JavaScript files, necessary for the application functioning, to the web-application `WEB-INF/classes` directory from the `api` sub-directory of the platform runtime directory.
- Copy the `pwc` directory of the JavaScript HTML5 client from the `bin` sub-directory of the platform runtime directory to `web` directory within a project.
- Configure the JDBC data source as a JNDI resource, specify its name, for example `test_db`. Configure the connection pool and JNDI resource. Copy the JAR file copy of the corresponding database driver to the directory available for the class loader.
- Configure the security domain (Realm) for working with built-in storage or external authentication service. For working with built-in storage, configure the JDBC security domain for working with `MTD_USERS` and `MTD_GROUPS` tables. See Security section.
- Configure the parameters of the web-application deployment in the `WEB-INF/web.xml` file. If necessary, configure settings in the configuration files specific for this application server.

- Deploy the web-application in the J2EE container/on the application server as a WAR archive or as a directory.

5.2. Configuring the deployment descriptor

To configure a web-application, edit the XML file of the `WEB-INF/web.xml` deployment descriptor.

- Set the initialization parameter `default-datasource` with name of the JNDI resource of the JDBC data source as its value, for example:

```
...
<context-param>
    <param-name>default-datasource</param-name>
    <param-value>test_db</param-value>
 </context-param>
...
```

- Set the initialization parameter `appelement` with name of the default application element as its value, for example:

```
...
<context-param>
    <param-name>appelement</param-name>
    <param-value>start.js</param-value>
 </context-param>
...
```

  This parameter is usefull while working with JavaSE client and Servlet container as a server. This parameter is useless while working with browser. * Add the default html file to be sent to browser, when browser requests an application url, for example:

```
...
  <welcome-file-list>
    <welcome-file>application-start.html</welcome-file>
  </welcome-file-list>
...
```

- Add the session event handler:

```
...
<listener>
    <listener-class>com.eas.server.httpservlet.PlatypusSessionsSynchronizer</listener-c
</listener>
...
```

- Add a reference to the data source resource:

```
...
<resource-ref>
    <description>Main database connection</description>
    <res-ref-name>test_db</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

```
...
```

- Add Platypus servlet configuration. In the `multipart/location` element specify the path to the folder for storing the downloaded files:

```
...
<servlet>
    <servlet-name>PlatypusServlet</servlet-name>
    <servlet-class>
com.eas.server.httpservlet.PlatypusHttpServlet
    </servlet-class>
    <multipart-config>
      <location>
          /home/user1/pub
      </location>
      <max-file-size>2097152</max-file-size>
      <max-request-size>2165824</max-request-size>
      <file-size-threshold>1048576</file-size-threshold>
    </multipart-config>
  </servlet>
<servlet-mapping>
    <servlet-name>PlatypusServlet</servlet-name>
    <url-pattern>/application/*</url-pattern>
</servlet-mapping>
...
```

- Configure the access and security constraints. For information on the security domain configuration, see the "Security" section.

After completing configuring, deploy the web-application as a folder or WAR archive in a servlet container or on the J2EE server.

5.3. Authentication configuration on a J2EE container

When an application works in a J2EE container, the platform runtime uses an authentication mechanism and roles provided by the container. To enable activation of the role access in this case, the user should pass the security constraint and get a role. To do this, configure a URL security constraint as a page containing Platypus forms, for which the access control based on roles should be provided. The following example shows the enabled security constraint for the `applicationStart.html` page; to get access to this page the user should be assigned any role:

```
...
<security-constraint>
  <web-resource-collection>
      <url-pattern>/application-start.html</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>*</role-name>
    </auth-constraint>
</security-constraint>
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/login.html</form-login-page>
```

```
      <form-error-page>/login-failed.html</form-error-page>
    </form-login-config>
</login-config>
<security-role>
    <role-name>*</role-name>
</security-role>
...
```

Specify the type of authentication, for example, `FORM` for authentication using HTML forms or `BASIC` for basic authentication according to RFC 2617.

Platypus Client supports `BASIC` authentication, so to its ensure correct operation, that particular type of authentication must be configured.

Configure the repository of information about users and J2EE container for using this repository. More detailed information on these settings is provided below.

When an application works in the J2EE container, you should use built-in web-server tools in addition to Platypus platform security constraints:

- Restrict access to application files over HTTP.
- Restrict access to application files for Platypus resources loader, which works over URL of the following type:

  application/resource/

  where resourcePath is the path to the resource in the Platypus application.

Configure access constraints in the `WEB-INF/web.xml` descriptor file.

The following example shows a portion of the `WEB-INF/web.xml` file. It contains constraints of access to files of the Platypus application, located in the `app` directory, except `public` sub-directory:

```
...
<security-constraint>
  <web-resource-collection>
      <!-- whitelist -->
      <web-resource-name />
      <url-pattern>/app/public/*</url-pattern>
      <url-pattern>/application/resource/public/*
      </url-pattern>
   </web-resource-collection>
   <!-- No auth constraint here for whitelist -->
</security-constraint>

<security-constraint>
  <web-resource-collection>
      <!-- everything other is restricted -->
      <web-resource-name />
      <url-pattern>/app/*</url-pattern>
      <url-pattern>/application/resource/*</url-pattern>
  </web-resource-collection>
  <auth-constraint />
</security-constraint>
...
```

5.4. J2EE Glassfish 3 configuration

Data source setup:

1. Copy the JAR file of the JDBC driver to the directory accessible to the class loader: `glassfish/domains/mydomain/lib`
2. Run the GlassFish administration console. To do this, start the server, for example, by using the asadmin utility. Then navigate to `http://hostname:4848` in browser, where hostname is the address of the Glassfish application server, for example: `http://localhost:4848`.
3. Create the JDBC connection pool: Resources → JDBC → JDBC Connection Pools → New, `javax.sql.ConnectionPoolDataSource` resource type, and also specify the database connection parameters: `url`, `username`, `password`.
4. Check the pool settings by clicking Ping.
5. Create the JNDI resource for the connection pool: Resources → JDBC Resources# New. Specify the name of the resource, for example`jdbc/main`, and specify the JDBC connection pool.

To configure Platypus for working with internal storage of user data or external authentication service:

- Configure J2EE Glassfish server for working with the security domain (Realm) in the external LDAP service.
- Configure user accounts.

To configure the Glassfish server:

- Add the security domain for Glassfish — to do this, change the server configuration (Configurations → Server-config → Security → Realms #New).

  Specify the name of the security domain, select the class name from the list or specify your own class: ** To use the built-in Platypus storage, specify the com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm class name. Configure properties, which are specific for this class:

Table 5.1. JDBCRealm security domain properties

`JAAS Context`	Identifier of the login module, JDBCRealm
`User Table`	ame of the user tables in the database, MTD_USERS
`User Name Column`	Name of the column in the user table for storing user names, USR_NAME
`Password Column`	Name of the column in the user table for storing password hashes, USR_PASSWD
`Group Table`	Name of the user group table, USR_GROUPS
`Group Name Column`	Name of the group name column in the user group table, GROUP_NAME
`Digest Algorithm`	Password hashing algorithm, MD5

- To use the external LDAP service (Active Directory, OpenLDAP, etc.) specify the com.sun.enterprise.security.auth.realm.ldap.LDAPReam class name; configure properties which are specific for this class.

Table 5.2. Basic and additional properties of the LDAPReam security domain

`JAAS Context`	Identifier of the login module, ldapRealm
`Directory`	ldap://server:389
`Base DN`	DC=ithit,DC=com
`Assign Groups`	platypus_default_role
`search-filter`	(&(objectClass=user)(sAMAccountName=%s))
`search-bind-password`	LDAP service password
`group-search-filter`	(&(objectClass=group)(member=%d))
`search-bind-dn`	ithit\user

Note

Set values of properties in accordance with the configuration of your LDAP server. `Assign Groups` property value, platypus_default_role group will be assigned to all users.

Configure JVM: Configurations → server-config → JVM Settings → Add JVM Option - by adding the following option: Djava.naming.referral=follow * In the `WEB-INF/glassfish-web.xml` file link roles to the security groups:

```
<glassfish-web-app error-url="">
...
  <context-root>/platypus</context-root>
  <security-role-mapping>
    <role-name>platypus_default_role</role-name>
    <group-name>default</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>role1</role-name>
    <group-name>role1</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>role2</role-name>
    <group-name>role2</group-name>
  </security-role-mapping>
..
</glassfish-web-app>
```

5.5. Apache Tomcat 8 configuration

Data source setup:

- Copy the corresponding JAR file of the JDBC driver to the directory accessible to the class loader: `/lib`, where CATALINA_HOME is a Apache Tomcat;

- Create the JNDI resource of the JDBC data source. Edit the `META-INF/context.xml` file of the web application by adding the data source resource:

```
...
<Resource name="jdbc/main" auth="Container" type="javax.sql.DataSource"
       maxActive="100" maxIdle="30" maxWait="10000"
       username="sa" password="te$tPwd" driverClassName="org.h2.Driver" url="jdbc:h2:t
...
```

If necessary, configure the connection pool by specifying parameters for removing and cleaning unused connections. * Configure the security domain. Edit the `META-INF/context.xml` file of the web application by adding the security domain. The example below shows configuring of the security domain for working with the built-in repository of user information:

```
...
<Realm  className="org.apache.catalina.realm.DataSourceRealm"
   dataSourceName="jdbc/TestDB"
   userTable="MTD_USERS" userNameCol="USR_NAME"   userCredCol="USR_PASSWD"
   userRoleTable="MTD_GROUPS" roleNameCol="GROUP_NAME" digest="MD5"/>
...
```

For the DataSourceRealm security domain, specify names of tables, columns and hashing algorithm for working with `MTD_USERS` # `MTD_GROUPS` tables.

If you want to use another authentication data repository, such as an external LDAP server, configure the appropriate type of security domain.

5.6. Logging level parameter values

Table 5.3. Logging levels (-loglevel)

`OFF`	Logging is disabled
`SEVERE`	Only messages related to serious problems, which prevent the application from normal starting, are displayed. This logging mode is useful for developers. Minimum level.
`WARNING`	Messages about possible problems are displayed. This logging mode will be useful for developers and system administrators.
`INFO`	Information messages are displayed. In this logging mode, messages, which are considerably significant and important for end users and system administrators, are displayed.
`CONFIG`	Messages about system configuration are displayed. This mode allows you to debug problems, which are associated with the basic PC configuration. For example, information about the

	processor, color depth, connected modules, etc. is shown.
`FINE`	Debugging information is displayed. This level will be interesting for software developers.
`FINER`	The deeper level of refining.
`FINEST`	The most detailed output of debugging information. Maximum level.
`ALL`	All messages are displayed.

5.7. J2EE Glassfish 3 configuration

Note

For more detailed information on Glassfish setup, refer to the documentation for this application server.

Data source setup:

1. Copy the JAR file of the JDBC driver to the directory accessible to the class loader: `glassfish/domains/mydomain/lib`
2. Run the GlassFish administration console. To do this, start the server, for example, by using the asadmin utility. Then navigate to `http://hostname:4848` in browser, where hostname is the address of the Glassfish application server, for example: `http://localhost:4848`.
3. Create the JDBC connection pool: Resources → JDBC → JDBC Connection Pools → New, `javax.sql.ConnectionPoolDataSource` resource type, and also specify the database connection parameters: `url`, `username`, `password`.
4. Check the pool settings by clicking Ping.
5. Create the JNDI resource for the connection pool: Resources → JDBC Resources# New. Specify the name of the resource, for example`jdbc/main`, and specify the JDBC connection pool.

To configure Platypus for working with internal storage of user data or external authentication service:

- Configure J2EE Glassfish server for working with the security domain (Realm) in the external LDAP service.
- Configure user accounts.

To configure the Glassfish server:

- Add the security domain for Glassfish — to do this, change the server configuration (Configurations → Server-config → Security → Realms #New).

  Specify the name of the security domain, select the class name from the list or specify your own class: ** To use the built-in Platypus storage, specify the com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm class name. Configure properties, which are specific for this class:

Table 5.4. JDBCRealm security domain properties

`JAAS Context`	Identifier of the login module, JDBCRealm
`User Table`	ame of the user tables in the database, MTD_USERS
`User Name Column`	Name of the column in the user table for storing user names, USR_NAME
`Password Column`	Name of the column in the user table for storing password hashes, USR_PASSWD
`Group Table`	Name of the user group table, USR_GROUPS
`Group Name Column`	Name of the group name column in the user group table, GROUP_NAME
`Digest Algorithm`	Password hashing algorithm, MD5

- To use the external LDAP service (Active Directory, OpenLDAP, etc.) specify the com.sun.enterprise.security.auth.realm.ldap.LDAPReam class name; configure properties which are specific for this class.

Table 5.5. Basic and additional properties of the LDAPReam security domain

`JAAS Context`	Identifier of the login module, ldapRealm
`Directory`	ldap://server:389
`Base DN`	DC=ithit,DC=com
`Assign Groups`	platypus_default_role
`search-filter`	(&(objectClass=user)(sAMAccountName=%s))
`search-bind-password`	LDAP service password
`group-search-filter`	(&(objectClass=group)(member=%d))
`search-bind-dn`	ithit\user

Note

Set values of properties in accordance with the configuration of your LDAP server. `Assign Groups` property value, platypus_default_role group will be assigned to all users.

Configure JVM: Configurations → server-config → JVM Settings → Add JVM Option - by adding the following option: Djava.naming.referral=follow * In the `WEB-INF/glassfish-web.xml` file link roles to the security groups:

```
<glassfish-web-app error-url="">
...
  <context-root>/platypus</context-root>
  <security-role-mapping>
```

```
    <role-name>platypus_default_role</role-name>
    <group-name>default</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>role1</role-name>
    <group-name>role1</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>role2</role-name>
    <group-name>role2</group-name>
  </security-role-mapping>
..
</glassfish-web-app>
```

5.8. Apache Tomcat 8 configuration

Note

For more detailed information on Apache Tomcat 8 setup, refer to the corresponding documentation.

Data source setup:

- Copy the corresponding JAR file of the JDBC driver to the directory accessible to the class loader: `/lib`, where CATALINA_HOME is a Apache Tomcat;
- Create the JNDI resource of the JDBC data source. Edit the `META-INF/context.xml` file of the web application by adding the data source resource:

```
...
<Resource name="jdbc/main" auth="Container" type="javax.sql.DataSource"
       maxActive="100" maxIdle="30" maxWait="10000"
       username="sa" password="te$tPwd" driverClassName="org.h2.Driver"
       url="jdbc:h2:tcp://localhost/~/h2db/test;schema=test"/>
...
```

If necessary, configure the connection pool by specifying parameters for removing and cleaning unused connections. * Configure the security domain. Edit the `META-INF/context.xml` file of the web application by adding the security domain. The example below shows configuring of the security domain for working with the built-in repository of user information:

```
...
<Realm  className="org.apache.catalina.realm.DataSourceRealm"
   dataSourceName="test_db"
   userTable="MTD_USERS" userNameCol="USR_NAME"   userCredCol="USR_PASSWD"
   userRoleTable="MTD_GROUPS" roleNameCol="GROUP_NAME" digest="MD5"/>
...
```

For the DataSourceRealm security domain, specify names of tables, columns and hashing algorithm for working with `MTD_USERS` # `MTD_GROUPS` tables.

If you want to use another authentication data repository, such as an external LDAP server, configure the appropriate type of security domain.