Platypus administration guide

Platypus administration guide

Table of Contents

List of Tables

Chapter 1. Platform’s installation

If the Platypus Platform 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
- Oracle JRE 7 or OpenJDK 7
- 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 client HTML5 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 7
- Glassfish 3.1.2
- JBoss AS7

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 components, perform the following actions:

1. Go to the `install` subdirectory of the Platypus distribution kit.
2. Run `install.bat` files for the Windows OS or perform the `sh install.sh` command for Linux.
3. Select the language of the installation program user interface.
4. Select the installation directory.
5. Select components to be installed.
    - The platform’s core components (required)
    - Java SE desktop client
    - Geospacial and maps components
    - Components required for operation of the standalone application server and in J2EE servlet containers and on the application servers
    - Components required for operation with COM/DCOM and OPC servers
6. Select settings of shortcuts for running the Platypus client in your system environment.

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

- `platypus/`
    - `bin/`
    - `lib/`
    - `etc/`
    - `run/`
    - `logs/`

1.3. Uninstallation guide

To delete the platform components from your computer, perform the following actions:

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

Chapter 2. Application management

2.1. Application structure and system tables

If an application is delivered as an archive file, unpack its contents to a your hard disk drive.

After unpacking, the directory will have the following structure:

- `projectDirectory/`
  - `app/`
  - `project.properties`
  - `private.properties`

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

The `app` directory contains application elements directories and files.

The `project.properties` and `private.properties` are project’s configuration files used by the IDE.

It is possible to deploy the application to the database `MTD_ENTITIES` table and run it from there.

System tables includes the following:

- `MTD_ENTITIES` — this table stores an executable application in the form of a tree.
- ‘MTD_USERS` # `MTD_GROUPS` — tables of users and their groups when working with built-in users’ space. For additional information, see the "Security" section.
- `MTD_VERSION` — this table stores information about the current version of the application data repository.

After deployment in the database each application element is represented as a single entry in the `MTD_ENTITIES` table.

Use the following command for application database mangement utility:

```
java -jar PlatypusDeploy.jar
```

where OPTIONS is a deployment or migration command and a set of its parameters.

For the description of commands related to the import deployment operation and migrations, see sections dedicated to deployment and database migrations.

Table 2.1. Database management common command line options

Parameter	Description
`-url DB_URL`	Database connection JDBC URL
`-dbuser DB_USER`	Username
`-dbpassword DB_PASSWORD`	Password
`-dbschema DB_SCHEMA`	Database scheme

2.2. Application database deployment and import

Database deployment is the process allows you to application to the database table `MTD_ENTITIES`.

To deploy or undeploy an application to/from a database use the database management utility command line with the one of the following options:

Table 2.2. Application deployment and import commands

Parameter	Description
`-initapp`	Checks and initializes application storage table in the database, if it is not initialized.
`-ap APP_PATH`	The path to the folder containing the application's project.
`-deploy`	Deploys application to the specified database.
`-undeploy`	Removes application from the specified database.
`-import`	Imports application from the database to the specified directory.

Example of the command for deploying an application to the database:

```
java -jar PlatypusDeploy.jar -deploy -ap ~/apps/testApp -url jdbc:oracle:thin:@serverHos
```

Example of the command for importing an application from the database:

```
java -jar PlatypusDeploy.jar -import -ap ~/apps/testApp -url jdbc:oracle:thin:@serverHos
```

2.3. Database migrations

Application database schema and initial data can be stored in as a consecutive migrations. Database migrations can be applied consistently to ensure the correct state of the application's database(s) schema and its initial data.

Each migration of the database schema or data is represented as a migration file with a name corresponding to the migration version, starting from 1. The next file name for each new migration is created by incrementing the current application version number stored in the `MTD_VERSION` table.

The `MTD_VERSION` table contains the following mandatory field:

Field	**Description**
`VERSION_VALUE`	The application’s current version number

The types of migration files are the following:

- Database metadata snapshot with the `.xdm` extension.
- A bundle of SQL commands for adding and/or changing service data with the `.batch`.

To create a migration files and apply them to a database use the command line with the the following options:

Table 2.3. Commands for creating and applying database migrations

Parameter	Description
`-initversioning`	Checks and inits the version storage table in the database, if it is not initialized.
`-migrations`	The path to the directory containing the migrations
`-snapshot`	Creates a new migration — a database metadata snapshot; the database version corresponds to this migration.
`-batch`	Creates a new empty batch migration; the version of the database corresponds to this migration.
`-clean`	Cleans the migration directory by removing migrations which are not applied.
`-getver`	Returns the current version of the database.
`-setver`	Sets the current database version in VERSION as a nonnegative integer.

When applying migrations to the database, only the latest snapshot and the snapshots which were created immediately before the `.batch` migration, other snapshot migration are ignored and can be removed using the `-clean` command.

The following example illustrates how to apply migrations to the database:

```
java -jar PlatypusDeploy.jar -url jdbc:oracle:thin:@serverHost:1521:adb -dbuser user1 -d
```

2.4. 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
`USR_FORM`	Default form application element name

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 2.4. 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.

2.5. Platypus applications deployment and database migrations

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 entirely in the database or on the hard drive in the form of a directory.

Migration process ensures maintaining and restoring configuration and original data in the database.

Platypus applications are provided by the following ways:

- On CD or DVD drive.
- Over the Internet as a zip archive file.

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/`
  - `src/`
  - `db/`
  - `platypus.xml`

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 `platypus.xml` configuration file, the `src` directory, containing application elements, `db` directory, containing database migrations, and `project.properties` and `private.properties` designer configuration files.

The database, in which the application is deployed, includes system tables and any number of worksheets. When deploying, system tables are created automatically.

System tables includes the following:

- `MTD_ENTITIES` — this table stores the executable application in the form of a tree.
- `MTD_USERS` # `MTD_GROUPS` — tables of users and their groups when working with built-in user space; for additional information, see the "Security" section.
- `MTD_VERSION` — this table stores information about the current version of the application data repository.

After deployment in the database each application element is represented as a single entry in the `MTD_ENTITIES` table.

The command for starting the deployment and database migration utility is as follows:

```
java -jar PlatypusDeploy.jar
```

where OPTIONS is a deployment or migration command and a set of its parameters.

For the description of commands related to the import deployment operation and migrations, see sections dedicated to deployment and database migrations.

Table 2.5. Deployment and migration commands parameters

Parameter	Description
`-ap APP_PATH`	Path to the folder containing the finished Platypus application. If this parameter is not specified, the current directory will be used.
`-dbuser DB_USER`	Username for authorization in the database.
`-dbpassword DB_PASSWORD`	User password for authorization in the database.
`-dbschema DB_SCHEMA`	Database scheme.

Note

You can also perform actions for deploying and updating applications using the Platypus Application Designer.

2.6. Application database deployment and import

Important

Before updating the application, make a backup copy of the operating database.

During the development, the application resides in a drive directory and it can be run for debugging and testing directly from the drive. For commercial usage, it is recommended to deploy the application to the database.

Table 2.6. Application deployment and import commands

Parameter	Description
`-deploy`	Deploys application from the drive to the database.
`-undeploy`	Deletes application from the database.
`-import`	Imports application from the database to the drive.

If settings of the connection to the database or the Platypus user name and password are not specified in options, information about the connection to the database will be read from the `platypus.xml` settings file of the application folder.

Example of the command for importing an application from the database:

```
java -jar PlatypusDeploy.jar -import -ap ~/apps/testApp -url jdbc:oracle:thin:@serverHos
```

Platypus Application Server and Platypus Client do not require restarting after updating the operating application.

2.7. Database migrations

Configuration of the Platypus application database is stored in the form of successive migrations, which are applied consistently to ensure the correct operation of schema and data updates in the database. Each change of the structure or business data is represented as a migration file with a name corresponding to the change version, starting from 1. The file name for each new migration is created by adding 1 to the maximum version number for the application. Two types of migration files are available:

- Database metadata snapshot with the `.xdm` extension
- A package of SQL commands for adding and/or changing service data with the `.batch`

Table 2.7. Commands for creating and applying database migrations

Parameter	Description
`-apply`	Applies a set of migrations to the database. The migrations with number greater than the current database version are applied.
`-undeploy`	Deletes application from the database.
`-snapshot`	Creates a new migration — a database metadata snapshot; the database version corresponds to this migration.
`-batch`	Creates a new empty batch migration; the version of the database corresponds to this migration.
`-clean`	Cleans the migration directory by removing migrations which are not applied.
`-getver`	Returns the current version of the database.
`-setver`	Sets the current database version in VERSION — a nonnegative integer.

If settings of the connection to the database or the Platypus user name and password are not specified in options, information about the connection to the database will be read from the `platypus.xml` settings file of the application folder.

When applying migrations to the database, only the latest snapshot of the database structure is used, as well as all snapshots which were created immediately before the packages of SQL commands; other migration files are ignored and can be removed using the `-clean` command.

Example of applying migrations to the database:

```
java -jar PlatypusDeploy.jar -apply -ap ~/apps/testApp
```

Chapter 3. Desktop client

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

The desktop client supports automatic updates via network.

3.1. Running desktop client

Startup scripts and Platypus client loader are located in the `/run` folder:

- `platypus.exe` — to run the client in Windows without creating the command prompt window with preliminary running the client update; this command ensures the correct functioning with UAC (User Account Control).
- `platypus.js` — a script to run the client in Windows without checking for updates.
- `platypus.sh` — to run the client in Linux.
- `startupdate.sh` — to run the client in Linux with preliminary running the client update.

If the desktop client is started successfully, it displays a dialog box for entering the user name and password and selecting a preconfigured connection to the server.

You can create and use several preconfigured connections to the database or to an app server. Your distribution kit can include a default preconfigured connection.

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.

Use the >>> and <<< buttons to display and hide the preconfigured connections panel.

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 the launch script 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  -cp Application.jar;com.eas.client.application.PlatypusClientApplication
```

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`	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 desktop client running command:

```
java -Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel -cp Application
```

Chapter 4. Platypus Application Server

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

4.1. Running server

To start the server application, you run the startup script for the server, located in the `/run` folder:

- `server.bat` is the script file to start the server in Windows.
- `server.sh` is the script file to start the server in Linux.

For running the server in production environment, it is recommended to configure it as an operating system service.

4.2. Command line options

The command for running the server is as follows:

```
java  -cp Server.jar; com.eas.server.ServerMain
```

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 application 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.
`-tasks` MODULES_NAME	The list of server modules (separated by comma without spaces) for processing incoming data and background tasks. The received(data)function should be defined for the incoming data processing module. The incoming data processing modules can have the `@stateless` annotation, or be without it. Background tasks modules should not have the `@stateless` annotation. When starting server, modules with this annotation will be skipped.

Parameter	**Description**
`-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 -cp Server.jar com.eas.server.ServerMain -iface 0.0.0.0:8500 -protocols 8500:platyp
```

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 user space.
- Database connecions are configrued as a JNDI resources.
- Application sever modules methods are avaliable via HTTP protocol.

> **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 plafrom’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` 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.
- Configure the JDBC data source as a JNDI resource, specify its name, for example `jdbc/main`. 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 and specify `url` as its name and path to Platypus application folder as its value; the path may be absolute or relative within the web-application; when the web-application is packed into a war file, the relative path will work if the server unpacks this archive when deploying the application; if this option is not specified, the application will be loaded from the database. In the following example, an absolute path is specified

```
...
<context-param>
    <param-name>url</param-name>
    <param-value>file:///home/Platypus/app</param-value>
 </context-param>
...
```

- Set the initialization parameter and specify `default-datasource` as its name and the 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>jdbc/main</param-value>
 </context-param>
...
```

- Add the session event handler:

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

- Add a reference to the data source resource:

```
...
<resource-ref>
    <description>Main database connection</description>
    <res-ref-name>jdbc/main</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>
    <load-on-startup>1</load-on-startup>
    <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 7 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:
  ...
  ```

  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 7 configuration

Note

For more detailed information on Apache Tomcat 7 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="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.