Standard Build Documentation

revision 1

6.7.2011

Nick Wellinghoff, updates provided by Mike Hovis and Adam Hampton

Purpose

Setting up a build environment is a tedious task that needs to be performed on every project deployment. The SailPoint services team provides a standard build environment to help you quickly get up and running at your customer. This system was designed with the following goals in mind.

* Automate effort of generating deployments for various environments such as development, testing, UAT and production.
* Reduce time frame for new team members to become familiar with project structure and customizations.
* Reduce likelihood of errors due to improper deployment of patches, efixes and customizations.
* Accelerate the software development process with useful methods and tools that make customizing SailPoint more pleasurable.
* Enable the SailPoint support team to quickly replicate customer environments

Please forward any errors or suggested improvements to this document to nick.wellinghoff@sailpoint.com . Thank you for helping us improve our documentation.

Table of Contents

[Components 3](#_Toc295393406)

[Getting Started with building environments 3](#_Toc295393407)

[The build.properties file 3](#_Toc295393408)

[Setting up environment specific properties files 5](#_Toc295393409)

[Folder Structure 7](#_Toc295393410)

[Executing the build 9](#_Toc295393411)

[Customizing the build process 9](#_Toc295393412)

[Build.custom.tomcat6.xml example 9](#_Toc295393413)

[How it works 11](#_Toc295393414)

[Using the build during development 12](#_Toc295393415)

[IDE Integration 12](#_Toc295393416)

[Integrating with the Eclipse IDE 13](#_Toc295393417)

[Revisiting build.properties 15](#_Toc295393418)

[Dev targets explained 15](#_Toc295393419)

[concurd 15](#_Toc295393420)

[createdb 15](#_Toc295393421)

[cycle 15](#_Toc295393422)

[dropdb 15](#_Toc295393423)

[dist 15](#_Toc295393424)

[importcycle 15](#_Toc295393425)

[importdynamic 15](#_Toc295393426)

[Getting the entire IIQ environment up in 10 minutes or less 16](#_Toc295393427)

# Components

The build system is separated into two parts. One part handles building of a war file for distribution that lays down the GA release, patches, efixes, environment specific configuration , custom java etc. and packages it into a war file for distribution to the customer system. The other part assists you during the software development process by automating tedious tasks like importing configuration, compiling java, creating and dropping the database, and cycling the application server. Each part is independent and the build files are laid out so you only have to understand components that you care about.

# Getting Started with building environments

The first step is to expand the build template archive you downloaded and setup the build.properties file found in the root directory. The first section, labeled required properties, is the minimum set of properties you will have to set to build.

### The build.properties file

**#required properties**

IIQVersion=5.1

IIQPatchLevel=p5

customer=AcmeBank

jdk.home=c:/Sun/Jdk

runCustomScripts=false

**#dev properties**

#make sure any app server specific env variables,like CATALINA\_HOME for tomcat, are set on your system

IIQHome=g:/tomcat/webapps/iiq

application.server.host=localhost

application.server.port=8080

application.server.start=G:/tomcat/bin/startup.bat

application.server.stop=G:/tomcat/bin/shutdown.bat

db.url=jdbc:mysql://localhost?useServerPrepStmts=true&tinyInt1isBit=true&useUnicode=true&characterEncoding=utf8

db.userid=root

db.password=password

db.driver=com.mysql.jdbc.Driver

iiq.path=/iiq

#type must be db2,mysql,oracle,sqlserver

db.type=mysql

#set this to true if you make test scripts that call targets with user warnings like drop db etc.

override.safety.prompts=false

#for export, the original install date string that we can use to determine new or changed objects

installDate=4/25/12 14:48:58 PM CDT

**#tomcat properties.** You only need to set these if you have a CATALINA\_HOME env var set

#NOTE: for tomcat 7+ the manager url is /manager/text for 6 and lower its just /manager

#You will also need to setup a manager-script user in your tomcat user config, check out

#http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Executing\_Manager\_Commands\_With\_Ant

manager.url=http://localhost:8080/manager/text

manager.login=tomcat

manager.pw=tomcat

#### properties explained

**IIQVersion -** Specify the base version of IIQ that you are building from. Ex. 4.0 or 5.2.

**IIQPatchLevel -** If you want to deploy a patch level specify what level with pX syntax. Ex. p1 or p6. If you are not on a patch level then leave this entry blank.

**IIQHome -** The home directory of the IIQ webapplication in your sandbox/dev environment.

**customer -**  The name of the client or project phase

**jdk.home -** The path on your system to the jdk you want to use to compile custom java.

**runCustomScripts -** (true/false) The build is not meant to be modified. You will notice the core build files are set to be read only. The main build has two hook points after file layout and after war creation where you can execute customized build scripts. This flag indicates if these customizations should be executed.

**application.server.host -** The IP address of your application server in your sandbox/dev environment

**application.server.port -** The port the application server is running on. 8080 is the Tomcat default.

**application.server.start -** Since there are so many different application servers we leave it to you to write a script that starts and stops the server, sets up JVM parameters etc. Many application servers already ship with these but you can specify which ones you want to use here. These scripts are used in dev targets that cycle the application server for you.

**application.server.stop -** same as above.

**db.url -** The JDBC url to your local database.

**db.userid -** db user that has create and drop schema privileges . Ex. root on Mysql. (Sandbox only)

**db.password** - The password for the root db user. (Sandbox only)

**db.driver** - The class of the JDBC driver to use for SQL connections

**iiq.path** - The context path to the IIQ web application. Usually /iiq or /identityiq

**db.type** - (db2/mysql/oracle/sqlserver) used to pick which database scripts to run.

**override.safety.prompts** - Certain dangerous build targets like dropdb will prompt the user for confirmation before executing. If you are using the build to make test cases you may want to turn off these prompts.

**manager.url -** URL to the tomcat manager script interface. Prior to version 6 the url is usually /manager but post version 6 its /manager/text.

**manager.login -** A user who has the manager-script role in the Tomcat manager application. For information on how to set this up check out. http://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html#Executing\_Manager\_Commands\_With\_Ant

**manager.pw -**  The password for the above account.

### Supporting multiple platforms (Windows/Linux/Unix) for different environments

If your environment has different operating systems for different stages of IIQ development – for example Windows for sandboxes and Linux for Test and Production servers then you will need to configure multiple “build.properties” files. The “build.properties” file described above loads the defaults for the build with respect to the path to java binaries, IIQ version and other details. In order to override those defaults on a per-server basis you can specify another properties file with properties that just apply to one server. Each server or host used in development and testing can override the settings in “build.properties” by using its own “build.properties.*hostname*“ file. For example if your host is named “sailsandbox” then the properties file unique to that server that server would be called “build.properties.sailsandbox”. The server’s properties file has the exact same format and fields as the “build.properties” file described in the previous section and only has to specify the fields that it wants to override with new values.

### Setting up environment specific properties files

Each environment has different logins, IPs and passwords. The build will automatically look for tokenized strings in your custom configuration xml and substitute the appropriate values per environment.

#### Setting the target variable by editing servers.properties

To setup this up you must add your machine hostname to the servers.properties file in the root build directory with the environment that the machine is supposed to use. This tells the build what environment you want to use. If your environments are sandbox, test, UAT and prod and you wanted to build using the settings for sandbox you would set your machine environment to "sandbox" (see servers.propertes).

#### <target>.iiq.properties

This file is the iiq.properties file you want to use for the target environment. For example, if your environments are sandbox, test, UAT and prod. You would have four files each containing the iiq.properties that know how to connect to the database server in that environment. This way you can support having a direct connection in sandbox and test while having a JNDI named connection in UAT and production.

Example

sandbox.iiq.properties

test.iiq.properties

UAT.iiq.properties

prod.iiq.properties

#### <target>.target.properties

This set of files contains name value pairs for token substitution during build time. You should have one file for each environment in your deployment.

Example

sandbox.target.properties

test.target.properties

UAT.target.properties

prod.target.properties

Each file is just a list of name value pairs. The build follows the convention that the keys follow the %%.\*%% pattern.

Example, you may have an AD application configuration that looks like this:

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE sailpoint PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<sailpoint>

<Application authoritative="true" connector="sailpoint.connector.ADLDAPConnector" featuresString="AUTHENTICATE, MANAGER\_LOOKUP, SEARCH, UNSTRUCTURED\_TARGETS" name="AD" profileClass="" type="Active Directory">

<Attributes>

<Map>

<entry key="password" value="mySuperSecrectProductionPassword"/>

<entry key="managerCorrelationFilter">

<value>

<Filter operation="EQ" property="DN" value="manager"/>

</value>

</entry>

<entry key="user" value="productionADuser"/>

<entry key="groupHierarchyAttribute" value="memberOf"/>

<entry key="port" value="3379"/>

<entry key="authorizationType" value="simple"/>

...

To support multiple environments you would substitute passwords, ports, etc. with keys that will go in your <target>.target.properties file.

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE sailpoint PUBLIC "sailpoint.dtd" "sailpoint.dtd">

<sailpoint>

<Application authoritative="true" connector="sailpoint.connector.ADLDAPConnector" featuresString="AUTHENTICATE, MANAGER\_LOOKUP, SEARCH, UNSTRUCTURED\_TARGETS" name="AD" profileClass="" type="Active Directory">

<Attributes>

<Map>

<entry key="password" value="%%AD\_PROXY\_PASSWORD%%"/>

<entry key="managerCorrelationFilter">

<value>

<Filter operation="EQ" property="DN" value="manager"/>

</value>

</entry>

<entry key="user" value="%%AD\_PROXY\_USER%%"/>

<entry key="groupHierarchyAttribute" value="memberOf"/>

<entry key="port" value="%%AD\_PORT%%"/>

<entry key="authorizationType" value="simple"/>

...

Then, for example, in the file “prod.target.properties” you would have:

#######################################################

# AD Connectors

#######################################################

%%AD\_HOST%%=example.com

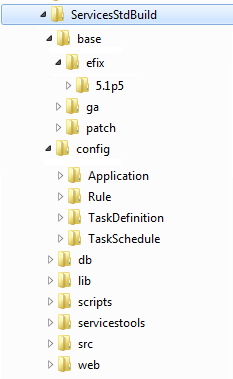
%%AD\_PORT%%=3379

%%AD\_PROXY\_USER%%=productionADuser

%%AD\_PROXY\_PASSWORD%%=mySuperSecrectProductionPassword

… and so on and so forth for each of your environments. Note: the password values should be IIQ encrypted strings.

## Folder Structure



This is the high level folder structure of the build. The top level directories should not be modified

**base -** Contains binaries distributed by SailPoint. You can download these from community.sailpoint.com

**efix -** Contains any efix archives sorted by directory name where the directory name follows the naming convention <version><patchlevel>. If there is no patch level then just the version number. Because efix solutions only work with the specific product version they were designed for you must make a unique directory for each version and patch level you want to build against. If the build does not find a efix directory for the version its building for it will fail with the error indicating so. In the above example there is a efix directory for 5.1p5.

**ga -** Contains the SailPoint GA release binary. You can have as many GA release binaries as you want to build against and the appropriate one will be selected using the values you set in build.properties.

Example: /base/ga/identityiq-5.1.zip

**patch -** Contains the SailPoint patch binaries. You can have as many patch binaries as you want to build against and the appropriate patch will be selected using the values you set in build.properties.

Example:/base/patch/identityiq-5.1p5.jar

NOTE: If you have your GA ZIP and/or patch JAR in a location outside of the SERI file structure, these locations can be specified with the GALocation and PatchLocation properties in build.properties. The properties should point to the directories containing the ZIP and JAR file.

**config -** Contains all SailPoint XML configuration sorted by folders where each sub directory is named by the type of top level SailPoint object it holds. In the provided example Application, Rule, TaskDefinition and TaskSchedule directories are shown. In general as you customize more object types you should add a directory to contain that object. While writing code, try to make the separation of object types as granular as possible such that it is easy to view all objects of a particular type. For example, instead of inlining a rule directly into a TaskDefinition, a reference to that rule should be created and the Rule itself would live in its own file in the Rule directory. Separate and encapsulate.

After the release of IIQ 6.0 the “LocalizedAttribute” object was introduced. This object holds local-language descriptions of Applications, Policies, and Bundles (roles). The presence of the “LocalizedAttribute” directory under config is to remind users to also export the description’s fields as “LocalizedAttribute” objects in addition to the Application, Policy, or Bundle XML.

**db -** Contains customized database scripts.

**lib -** Contains libraries used by the build process.

**scripts -** With the exception of the master build.xml file in the root directory, all other build files are contained in this directory. Shipped and supported build files are read-only and follow the name convention build.\*.xml. If you customize the build process you must declare your customizations in build files that follow the naming convention build.custom.\*.xml. These files adhere to an interface which is discussed in detail in the **Customizing the build process** section of this document.

**servicestools -** This folder may or may not be included in your version of the build archive. It contains the source code and an ANT project to build the services-tools.jar which is placed in the lib directory of the build.

**src -** Contains all of your custom java. Note this java will be placed in a jar file which will be placed in the main WEB-INF/lib directory. You should NOT "clone and own" SailPoint shipped classes in this area. Since they will be placed in the classpath at the same level as the shipped classes you may get behavior you do not expect. If you absolutely must modify a core class you will have to define a build.custom.\*.xml file to handle layout of these files as you are basically defining your own efix. By default the build will not play nice with this practice.

**web -** Contains content that will be directly overlaid on the IIQ folder structure. Examples include custom graphics/branding, xhtml, jsp and custom message catalogs etc.

## Executing the build

Once you have performed all the steps in the previous sections you are ready to build. Open a command prompt and type.

c:\myProject>build.bat war

Note: The build.bat file just calls the version of Ant that is included with the build. We use Ant version 1.8.2. If you want to use the version of Ant resident on your system, change your command to:

c:\myProject>ant war

This will make a build\deploy directory under your root project directory and the war file will be placed in it. You are now ready to distribute to this file to your application server manually or by using the post war hook point. This concept is covered in the next section.

# Customizing the build process

The build process is designed to handle most common use cases and should not need to be heavily customized. If you feel a feature is missing and should be part of the standardized build, please write an example and submit it to the SailPoint services team and we will include it. Items of particular interest are areas where we could improve integrating with other standard build solutions.

That being said, there will be scenarios where you need to have customized scripts running during build time. The goal is to explicitly isolate these customizations. So in the event of a support case or project hand off, a new resource can easily focus in on the area of the process you have modified.

The main build scripts are read only files. If these files are overridden the build will no longer be supported by SailPoint Services in the event of failure. If you want to introduce a custom script and targets you will have to make a new file following the naming convention of build.custom.\*.xml . Place this file in the /scripts directory. The following example is included in the build for your reference.

## Build.custom.tomcat6.xml example

Customer Acme has chosen Tomcat 6 as their container for the application. It happens to be the case that Tomcat 6 ships with libraries that conflict with the IIQ standard install. We will make a custom build script that will handle removing the conflicting libraries to make our war file Tomcat 6 ready.

<?xml version="1.0" encoding="UTF-8"?>

<project name="services.standard.build.custom.tomcat6.xml">

<target name="post.expansion.hook">

<!-- This file messes up tomcat 6 because it includes its own el- api.jar -->

<move failonerror="false" file="${build.iiqBinaryExtract}/WEB-INF/lib/el- api.jar" tofile="${build.iiqBinaryExtract}/WEB-INF/lib/el- api.jar.disabled"/>

<!-- disable database driver as one is provided by application server-->

<!--<move failonerror="false" file="${build.iiq.war.dir}/WEB-INF/lib/sqljdbc-1.2.jar" tofile="${build.iiq.war.dir}/WEB-INF/lib/sqljdbc-1.2.jar.disabled"/>-->

</target>

<target name="post.war.hook"/>

<target name="clean"/>

</project>

The custom script must define the post.expansion.hook, post.war.hook and clean targets in order for the main build script to interface with it. You can of course define as many custom targets as you want in this file but these three targets must be the entry points to them. In this example we "disable" the el-api.jar that ships with IIQ as it conflicts with Tomcat 6's el-api.jar.

## How it works

Here are the primary targets from the main build script.

<target name="main"

depends="init-properties" description="makes complete iiq distribution directory">

<!-- laydown initial binaries and custom static web content -->

<antcall inheritall="true" target="prepBinaries"/>

<!-- search for custom tasks to perform on the files that will be included in the web application folder -->

<if>

<equals arg1="${runCustomScripts}" arg2="true"/>

<then>

<subant target="post.expansion.hook" inheritall="true">

<fileset dir="." includes="scripts/build.custom.\*.xml"/>

</subant>

</then>

</if>

<!-- compile custom java artifacts and put them a in jar -->

<antcall inheritall="true" target="includeCustomJar"/>

<!-- compile custom configuration, apply env tokens and setup init files-->

<antcall inheritall="true" target="prepareCustomConfig"/>

</target>

<target name="war" depends="main" description="Makes war file from main target results and applies custom post war custom scripts">

<!-- package a war file for deployment -->

<war destfile="build/deploy/identitiyiq.war">

<fileset dir="${build.iiqBinaryExtract}"/>

</war>

<if>

<equals arg1="${runCustomScripts}" arg2="true"/>

<then>

<subant target="post.war.hook" inheritall="true" failonerror="false">

<fileset dir="." includes="scripts/build.custom.\*.xml"/>

</subant>

</then>

</if>

</target>

There are two main hook points. If you have the "runCustomScripts" property set to true in build.properties the build will attempt to search for and execute all scripts named /scripts/build.custom.\*.xml that adhere to the interface.

**post.expansion.hook -** If there is a need to include any file artifacts like generated configuration or anything where the directory/file structure of the war distribution needs to be modified use this hook. In the example we use it to rename a jar file. A more interesting case would be a program that generates Application configuration XML following some sort of standard. Or you could do code validation and testing.

**post.war.hook -** This target will get called after the war file is generated. Often different application servers need the file in a specialized format. For example, you may wish to repackage the war file as a ear.

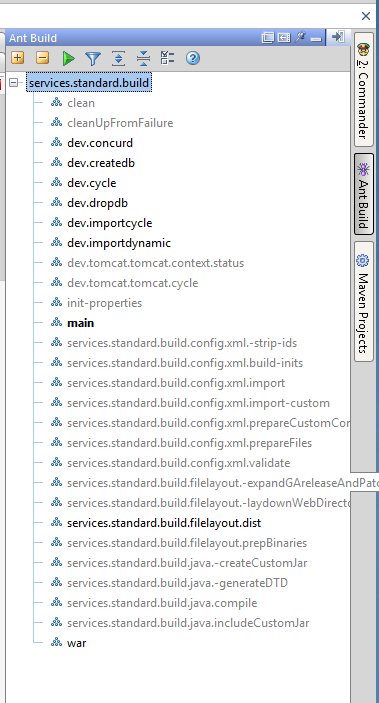
**clean -** If your script generates any artifacts that need to be cleaned up this target will be called by the main build clean process.

# Using the build during development

The second part of the build system is designed to make your life easier as a SailPoint Consultant during the development phase. If you are currently not leveraging an IDE to manage, debug and test your SailPoint configuration base we highly recommend your try it out! The most popular options are IntelliJ IDEA Community Edition or Eclipse IDE platforms, but any IDE that has built in Ant integration will do. In this section we will explore some of the tools provided. This portfolio of tools will be continually expanding so make sure you have downloaded the latest build archive from community.sailpoint.com.

## IDE Integration

Use your IDE's Ant integration feature and import the main build.xml to your project. You can now execute targets directly from your UI.

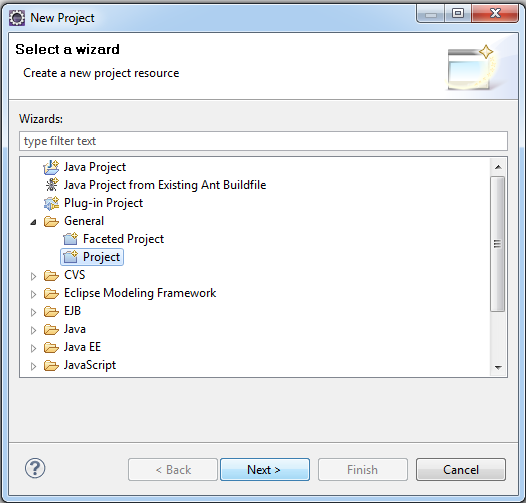


Screen shot from IntelliJ IDEA Ant integration.

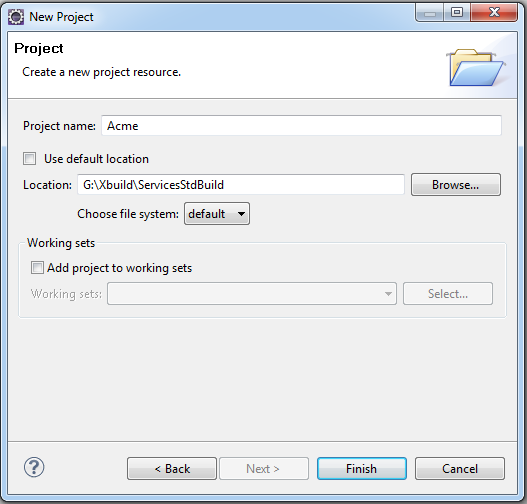
Most targets useful during development will be prefixed with "dev".

## Integrating with the Eclipse IDE

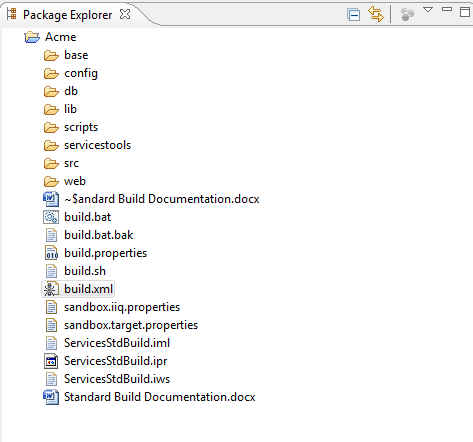
Create a new project and select "General", then "Project".



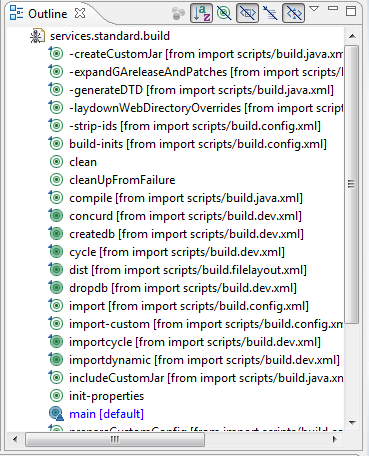
Uncheck "Use default location" and select the directory where you extracted the build archive.



Finish the project and expand the project folder. Double click on the build.xml file



You should now see the targets show up in the Ant integration pane.



## Revisiting build.properties

In order to leverage most of the dev targets, you will need to make sure all of the properties in build.properties #dev and #tomcat sections are correct. The build includes special support for Tomcat if you are using IIQ 5.2 or greater. To make use of these targets please make sure you have the CATALINA\_HOME environment variable set. For explanations on what each property is, refer the "the build.properties file" section in the first part of this document.

## Dev targets explained

### createdb

Depends on the build.properties file having a database account setup that has schema creation and drop privileges. This will setup the IIQ schema and apply any patch updates.

### cycle

Depends on the application.server.start and stop properties being set. This will cycle your application server and reload all web applications. If you are using Tomcat and you have the CATALINA\_HOME environment variable set this target will just reload the IIQ application and not the entire server.

### dropdb

Drops the IIQ database.

### dist

Copies the entire expanded war content to your application server webapps directory.

### importcycle

Runs through the entire build process, validates and imports all custom xml, compiled java and static web content and cycles the server. Useful while developing custom java.

### importdynamic

Imports all content that does not require an application reload; Custom XML, static web content etc. Useful for developing rules , workflow and branding.

### export

Exports objects specified within objectsToExport.properties (this will now be generated – edit Rule-OutputCustomObjectFile.xml in scripts if you need to add more objects to ignore or export – the variable name of ignored object classes is listOfIgnoredClasses.) from your IIQHome respository to build/export so that you don’t manually have to copy and paste xml from console-exported files to your build environment. Edit the property file to include all types of objects you want to export, as well as the names of the objects for each type.

## Getting the entire IIQ environment up in 10 minutes or less

This section assumes the environment already has a database server, application server and a community.sailpoint.com account already setup.

1) Go to community.sailpoint.com and download a GA release and patch level binary from the download section. Place the GA release zip in the base/ga directory and the patch zip in the /base/patch directory.

2)Set the product version and patch level in the build.properties file that corresponds to the zip files you just downloaded. While you are there input your application server settings and database account settings. If you are using the standard MySQL and Tomcat 7 dev environment you can leave most properties set to their default values.

3) Open a command line and change to the build directory. Type

build createdb

build importandcycle

4)Open a web browser and navigate to the IIQ url (http://localhost:8080/iiq)

5)You are up and running.