

atg

Table of contents

1 Welcome to the "Elements" Java Server Module Library.....	2
2 Install.....	3
2.1 Install.....	3
3 Configure.....	6
3.1 Configure.....	6
4 Run.....	12
4.1 Run.....	12
5 Reference.....	17
5.1 Reference.....	17
5.2 Types.....	18
6 All.....	21

1. Welcome to the "Elements" Java Server Module Library

The aim of this library is to provide a framework that automates the build from source code and reference data, and promotion of updates from one environment to another for server-side Java applications (principally J2EE compliant application server based applications).

The library contains the means to build applications from source code, package build artifacts such as modules and J2EE enterprise archives, and deploy these updates to target environments consisting of both web and application server tiers. This automation also exports data from a source database schema, packages it, deploys and imports it to target database schemas using one standardized end-to-end process.

Over-arching both of these goals the library provides process automation for the coordination of both code and data deployments including integrating continuous integration.

What is it?

This is a library of command modules that build on the ControlTier automation base types and establish an end-to-end data build and deployment process. This process is defined in terms of command workflows. The figure below describes three primary workflows:

1. **Coordinated Build and Deploy:** This workflow defines the combined end-to-end code module and reference data management process and a single command to execute it called "BuildAndDeploy".
2. **Code and Data Build:** This workflow is responsible for code module builds and exporting data from the master database, packaging them registering them with the repository.
3. **Code and Data Deploy:** This workflow manages the distribution and installation of the application code and reference data packages, installing modules importing the export files.

Why use it?

This library builds on the ControlTier service provisioning platform to offer a number of notable features for organizations where data migration and promotion is an important part of the application lifecycle and is a frequent and critical process:

- *End-to-end automation:* This framework ties together many individual procedures that are often run independently. Building on the ControlTier base types, this Library offers a single command that: builds code, exports data, packages and versions them, manages package dependencies and deploys to multiple hosts, installing the application and importing data from environment to environment.
- *Rollback:* All distributions are packaged and versioned. You can rollback to any previous version.

- *Task delegation and self-service*: You end up with schedulable "jobs" letting a less knowledgeable person run the process on demand or at defined periods.
- *Standardized operational interface*: End users can rely on simple and predictable commands no matter what environment, database platform, schema or application.
- *Reporting*: Every deployment is logged. Reports show who deployed what where and when. The full output of the job is saved and can be used for auditing later.
- *Graphical interfaces*: All operational tasks can be done via a web-based graphical interface. ControlTier's JobCenter is used to run commands while Workbench can be used to review the current deployment environment dependencies.
- *Security*: Using the ControlTier access control infrastructure, you can control who updates what when and where.
- *Extendability*: The library offers a working end to end process but it is not monolithic. You can override various parts of the process via sub-typing.

Getting started

You can start using the this library by following the steps of the documentation pages listed below:

1. [Install](#): Describes how to download and install the library
2. [Configure](#): Describes how to configure the library to define new "BuildAndDeploy" jobs
3. [Run](#): Explains how to run the job either via the graphical JobCenter application or by command line.

[Next: Install #](#)

2. Install

2.1. Install

Overview

This document describes the installation steps necessary to use the library.

2.1.1. Step #0: Prepare for installation

This module library has been tested to support a broad range of platform technologies and application topologies, far beyond the scope of this documentation to cover comprehensively. In general, you can assume that the library supports all major release of the [Apache Tomcat servlet container](#) and the [JBoss application server](#) deployed both Linux/Unix and Windows servers. Additionally, a number of common database servers (e.g. MsSQL and Hypersonic SQL) are supported.

In order to document the journey to a working solution from a "standing start", the following assumptions are made:

- The goal is to deploy Sun's Duke's Bank J2EE sample application (from their [J2EE 1.4 Tutorial](#)) to a single box.
- Duke's Bank is to be built and deployed to the JBoss application server running on either a Windows or Linux system.
- The Duke's Bank data will be stored in an stand-alone Hypersonic SQL database system (external to JBoss).

Specifically, then, for this demonstration of the module library the following software versions are required:

- JBoss 4.0.3sp1
- Hypersonic SQL Database 1.8.0.9
- Ant 1.7.0
- Cruise Control 2.7.1
- CentOS release 4.5 (Final), or Redhat Enterprise Linux 4 update 2 or better, or ...
- Windows XP Professional SP2, or Windows 2003 Server R2 (and possibly other Windows versions though they haven't been tested)

In addition, the availability of the following tools in your general environment is assumed:

- JDK 1.5.0
- Subversion 1.1.4, or later

Note:

Assuming you wish to configure the Duke's Bank demonstration, it is assumed that the machine(s) you choose to install on have access to the Internet to checkout demonstration source using Subversion from Sourceforge.

Note:

In order to run the entire set of services of the Duke's Bank demonstration on a single box (or virtual machine instance), you'll need around 2GB of disk space and 1024MB of memory. A three to six box configuration with separate server, build, and development and staging application server and database server systems can be run with virtual machines assigned 256-512MB of memory each and at 500MB of free disk space (1.5GB on the ControlTier server).

2.1.2. Step #1: Install the ControlTier platform software

This library assumes you have installed the latest stable release of the ControlTier 3.2 platform software on a designated server host and one or more client hosts. Refer to the [general installation procedures](#) for more info.

You will need to know the URL to the Workbench and JobCenter applications.

Note:

The module library has been tested using JDK 1.5.0 with the ControlTier framework

Note:

If you plan a multi-box installation be sure up update the "server.tomcat.hostname" and "client.hostname" properties in the "default.properties" command-line installer configuration file to appropriate network host names (these are also settable options in the GUI installer).

2.1.3. Step #2: Choose or create a project

All work is done within the context of a "project". You may already have a project in mind, or you may wish to create a new one just for the use of this library.

Note:

If you are new to ControlTier and are not sure about projects see the [Projects section in the ProjectBuilder tutorial](#).

1. Navigate to the Admin page. (eg., go to the URL:
<http://localhost:8080/itnav/do/menu/Admin>
2. Press the "Create Project" button. (eg., go to the URL:
<http://localhost:8080/itnav/do/projects/Input>

Note:

If you are following this documentation verbatim in order to configure the Duke's Bank demonstration, create a project called "DukesBank" (JBoss 4.0 "Duke's Bank" sample application project).

3. Fill out the form and press "Create" button. It takes a few minutes for the new project to be created.

Note:

If you are building a multi-box environment, this is the moment to register your client nodes with the new project using "depot-setup"

2.1.4. Step #3: Download the library archive

Binary distributions of the module library can be found in the "File Releases" section of the [ModuleForge Download](#) page on Sourceforge. The package is called "Elements Module Library" and will be named something like: elements-x.y-seed.jar where "x.y" denotes the version.

You must download at least version 3.0 of the module library.

2.1.5. Step #4: Load the library archive

Once you have chosen the desired project, you can load the library into that project.

Be sure you have already logged into Workbench and selected the desired project where you want the library loaded. If you just created a new project, you are all ready.

1. Navigate to the Admin page. (eg., go to the URL:
<http://localhost:8080/itnav/do/menu/Admin>
2. Press the "Import Seed" button. (eg., go to the URL:
<http://localhost:8080/itnav/do/projects/ImportSeedInput>
3. Locate and select the content-seed-x.y.jar file in the file chooser. This is the same file you downloaded in step #2.
4. Press the "Import" button.

Note:

It takes a few minutes for Workbench to load the library.

Once the library has been installed into your chosen project the next step is to configure these modules for use.

[Next: Configure #](#)

3. Configure

3.1. Configure

Overview

This document describes how to configure a project to use the content library.

The diagram below illustrates the configuration is driven by a `defaults.xml` file. This file is used as input by a command `generate-objects` which in conjunction with template files, produces two output files: `object.xml` and `job.xml`.

Note:

Be sure you have already installed the ControlTier software, chosen a project and loaded the library archive. See the [Install](#) page for instructions.

3.1.1. Step #1: Edit defaults.xml

The `defaults.xml` file contains all the essential environment-specific information needed by


```

        <!-- The default location of the installation directory for the
Duke's bank -->
        <!-- application source files and packages:
-->
        <dukesbankroot>${user.home}/dukesbank</dukesbankroot>

        <!-- The out-of-the box Duke's Bank demonstration can be configured
to run on -->
        <!-- from one to six systems. One system must be designated the
ControlTier -->
        <!-- server. This is the system where Workbench and Jobcenter run.
Another -->
        <!-- system is designated as the build box where CruiseControl and
the Ant -->
        <!-- based build are run.
-->
        <!--
-->
        <!-- One or two boxes are assigned to the development environment.
Builds are -->
        <!-- run here, and the development JBoss instance and "source"
HSQLDB database -->
        <!-- instance are deployed here. A fifth and possibly sixth system
host the -->
        <!-- staging environment which includes both JBoss and HSQLDB
instances. -->
        <!--
-->
        <!-- Note that all these systems can be the same box in which case
separate -->
        <!-- JBoss and HSQLDB server instances are started on separate
ports. -->

        <server>
            <node>${defaults.default.node}</node>
        </server>
        <development>
            <buildserver>
                <!-- The default copy of the JBoss 4.0 modified
Duke's Bank source code is kept under Subversion on Sourceforge: -->
                <scmconnection>https://moduleforge.svn.sourceforge.net/svnroot/moduleforge/elements/bra
                <node>${defaults.default.node}</node>
            </buildserver>
            <applicationserver>
                <node>${defaults.default.node}</node>
            </applicationserver>
            <databaseserver>
                <node>${defaults.default.node}</node>
            </databaseserver>
        </development>
        <staging>
            <applicationserver>
                <node>${defaults.default.node}</node>

```

atg

```
        </applicationserver>
        <databaseserver>
            <node>${defaults.default.node}</node>
        </databaseserver>
    </staging>
</defaults>
```

You are at liberty to change any of the six box names to systems that make sense in your environment, thereby ending up with the sample Duke's Bank development and staging environments deployed to as many as six and as few as a single system.

Note:

A copy of the defaults.xml template file can be found in the WebDAV under your project (substituting a host name for "localhost" if the ControlTier server is deployed remotely):
"http://localhost:8080/webdav/project/modules/ElementsProjectBuilder/templates/defaults.xml" ... where *project* is "DukesBank" if you are following the demonstration setup.

3.1.2. Step #2: Configure library objects

Register and install an ElementsProjectBuilder object:

```
ctl -p project -m Deployment -c Register -- \
    -name name -type ElementsProjectBuilder \
    -basedir $CTIER_ROOT/src/project -installroot
$CTIER_ROOT/target/project \
    -install
```

... or, specifically for the Duke's Bank demonstration:

```
$ ctl -p DukesBank -m Deployment -c Register -- -name elements -type
ElementsProjectBuilder -basedir $CTIER_ROOT/src/elements -installroot
$CTIER_ROOT/target/elements -install
Checking for existing object, (ElementsProjectBuilder) elements, in
project, 'DukesBank'...
Registered new object.
.
.
.
[command.timer.Deployment.Register: 3.096 sec]

C:\>ctl -p DukesBank -m Deployment -c Register -- -name elements -type
ElementsProjectBuilder -basedir %CTIER_ROOT%\src\elements -installroot
%CTIER_ROOT%\target\elements -install
.
.
.
[command.timer.Deployment.Register: 2.359 sec]
```

Copy the defaults.xml you created in [Step #1](#) to `$CTIER_ROOT/src/project/defaults.xml` and run the generate-objects command:

```
ctl -p project -t ElementsProjectBuilder -o name -c generate-objects --
-defaults $CTIER_ROOT/src/project/defaults.xml -upload
```

... or, specifically for the Duke's Bank demonstration (using different defaults XML files for Linux and Windows):

```
$ ctl -p DukesBank -t ElementsProjectBuilder -o elements -c
generate-objects -- -defaults
$CTL_BASE/depots/DukesBank/lib/ant/modules/ElementsProjectBuilder/templates/defaults.li
-upload
.
.
.
[command.timer.generate-objects: 0.156 sec]
generate-objects completed. execution time: 0.156 sec.
```

After this command successfully completes, a new set of objects will be loaded into the ControlTier repository. You can view them via ElementsProjectBuilder's find-objects command:

```
ctl -p project -t ElementsProjectBuilder -o name -c find-objects -- \
-name aName
```

Before you can run any commands, it is necessary to deploy the objects. This is done via the AntDepo command, `ctl-depot`. Run the following installation command on all boxes in your configuration:

```
ctl-depot -p project -a install
```

... or, if you're following the Duke's Bank demonstration setup instructions, more specifically (for a single box example):

```
$ ctl-depot -p DukesBank -a install
"Install" command running for object: (Site) developmentCI
"Install" command running for object: (HsqldbRdb) staging
"Install" command running for object: (Updater) development
"Install" command running for object: (HsqldbRdb) development
"Install" command running for object: (Site) developmentApplicationServer
"Install" command running for object: (ElementsProjectBuilder) elements
"Install" command running for object: (Site) stagingDatabaseServer
"Install" command running for object: (CruiseControl) development
"Install" command running for object: (AntBuilder) development
"Install" command running for object: (Site) stagingApplicationServer
"Install" command running for object: (Site) developmentDatabaseServer
"Install" command running for object: (HsqldbRdbExportBuilder) development
"Install" command running for object: (Updater) staging
"Install" command running for object: (JBossServer) development
"Install" command running for object: (JBossServer) staging
```

3.1.3. Step #3: Upload job definition

The `generate-objects` command run in [Step #2](#) will have produced a `job.xml` file with a filename `aName-job.xml`. This file can be used to define a new job in the JobCenter application.

1. Login to JobCenter (e.g, go to URL: <http://localhost:9090/jobcenter/menu/index>)
2. Press the "Create a new Job..." button
3. Press the "Upload job.xml" button
4. Locate and select the file, `aName-job.xml`, produced by `generate-objects` in the file chooser
5. Press "Save" button

The new job will be listed on the home page of JobCenter.

3.1.4. Step #4: Upload packages to the repository

The solution library also manages all the platform (3rd party) software packages needed to establish environments. The only assumptions are that you have a compatible OS image at your disposal, a user account with the ControlTier client installed and available, and sufficient disk space to deploy the platform and application software.

Note:

The Duke's Bank demonstration assumes you have a system with Java and Subversion pre-installed.

The general method of adding 3rd party packages to the ControlTier repository is by uploading them using Workbench:

If your intention is to run the Duke's Bank demonstration and you have generated and configured the example library objects mentioned above, then upload the following packages into the ControlTier repository via Workbench (i.e. select package objects and upload one at a time):

- The Zip archive of the binary distribution of Ant 1.7.0 (<http://ant.apache.org/bindownload.cgi>).
- The Zip archive of the binary distribution of Maven 2.0.9 (<http://maven.apache.org/download.html>).
- The Zip archive of the binary distribution of CruiseControl 2.7.1 (http://sourceforge.net/project/showfiles.php?group_id=23523&package_id=16338).
- The Zip version of the JBoss 4.0.3SP1 subscription or community edition (<http://labs.jboss.com/jbossas/downloads/>).
- The Zip archive of HSQLDB 1.8.0.9

http://sourceforge.net/project/showfiles.php?group_id=23316&package_id=16653).

- The initial version of the Duke's Bank data package from Moduleforge (<http://moduleforge.svn.sourceforge.net/viewvc/moduleforge/elements/branches/3.0/demo/DukesBank/d>

Next: Run #

4. Run

4.1. Run

Overview

This section describes how to prepare for and run a code module and data BuildAndDeploy workflow job or command and is pertinent to users responsible for releasing content changes from the source code repository and reference database to targeted deployment environments.

There are two interfaces available to execute the BuildAndDeploy commands:

- Using Jobcenter, the web-based graphical application,
- Using AntDepo's `ctl` shell command

Instructions for running the BuildAndDeploy command using either interface are explained below.

4.1.1. Preparing to run the Duke's Bank demonstration

In general, there are some one-off package installation and builder/service configuration commands to be run before normal ("day-to-day" build and deployment) operations can commence. In particular (regarding the Duke's Bank demonstration that has been installed and configured in the previous tabs of this site) execute the following commands from the Ctl client designated as the server system:

Note:

While instructions for these one-off commands are given in terms of using the Ctl shell command it is quite possible to configure these (and any other) commands to execute from Jobcenter too.

- Checkout the Duke's Bank application source code to the build box:

Note:

By default the source is checked out from Sourceforge's Subversion repository. If you would like to be able to make updates, transfer the source into your own repository and modify the development AntBuilder and MavenBuilder's BuilderScmConnection resource setting via ProjectBuilder or Workbench.

atg

```
$ ctl -p DukesBank -t AntBuilder -o development -c scmCheckout
scmCheckout parameters:
{basedir="/home/anthony/dukesbank/development/src",
connection="https://moduleforge.svn.sourceforge.net/svnroot/moduleforge/elements/bra
module="", label="" }
.
.
.
Checked out revision 635.
[command.timer.Builder.scmCheckout: 1:22.116 sec]
```

... OR:

```
$ ctl -p DukesBank -t MavenBuilder -o development -c scmCheckout
scmCheckout parameters:
{basedir="/home/jboss/dukesbank/development/src",
connection="https://moduleforge.svn.sourceforge.net/svnroot/moduleforge/elements/bra
module="", label="" }
.
.
.
Checked out revision 635.
[command.timer.Builder.scmCheckout: 1.653 sec]
```

- Install and configure the Cruise Control based continuous integration environment by issuing the following command from your ControlTier server box:

```
$ ctl -p DukesBank -t Site -o developmentCI -c Deploy
Start: "Run the coordinated deployment cycle across the configured
Sites." commands: dispatchCmd
begin workflow command (1/1) -> "dispatchCmd -command Deploy
-resourcename .* -resourcetype [^\.]*" ...
dispatching command: "Deploy " to: (CruiseControl) development ...
dispatching to object: development [CruiseControl] -> "Deploy "
.
.
.
command completed successfully. Execution time: 1:12.122 sec
dispatched command: Deploy completed for: (CruiseControl) development
end workflow command (1/1) -> "dispatchCmd -command Deploy -resourcename
.* -resourcetype [^\.]*"
[command.timer: 1:13.990 sec]
Completed: execution time: 1:13.990 sec
```

The CruiseControl instance should be available from build box at <http://localhost:8081> or similar (depending on your node setup). In addition you'll find a "boot.log" and "cruisecontrol.log" in the CruiseControl installation directory.

Note:

As soon as CruiseControl comes up for the first time it will kick-off initial Ant and Maven based builds (sequentially) and put the resultant package into the ControlTier server's package repository.

Note:

Deploying CruiseControl also installs the package dependencies (JBoss, Ant and Maven) required by the AntBuilder and MavenBuilder objects.

- As a general rule, you can find an object's configuration properties (for example, in this case, the Cruisecontrol installation directory - ccDir) as follows:

```
$ ctl -p DukesBank -t CruiseControl -o development -c Properties
[MULTI_LINE]
# development [CruiseControl] #

Duke's Bank continuous integration server

## Attributes ##

* basedir: "/home/anthony/dukesbank/development/src"
* ccDir: "/home/anthony/dukesbank/development/cruisecontrol-bin-2.7.1"
* interval: "60"
* javaHome: "${env.JAVA_HOME}"
* jmxport: "8001"
* path: "/bin:/usr/bin"
* rmiport: "1100"
* targetdir:
"/home/anthony/dukesbank/development/src/j2eetutorial14/examples/bank"
* webport: "8081"
.
.
.
```

- Install and configure the development and staging JBoss and HSQLDB database instances executing these commands from the ControlTier server node:

```
$ ctl -p DukesBank -t Updater -o development -c Deploy
Start: "Run the coordinated deployment cycle across the configured
Sites." commands: dispatchCmd
begin workflow command (1/1) -> "dispatchCmd -command Deploy
-resourcename .* -resourcetype [^\.]*Site" ...
dispatching command: "Deploy " to: (Site) developmentDatabaseServer,
(Site) developmentApplicationServer ...
.
.
.
Completed: execution time: 33.946 sec
dispatched command: Deploy completed for: (Site)
developmentDatabaseServer, (Site) developmentApplicationServer
end workflow command (1/1) -> "dispatchCmd -command Deploy -resourcename
.* -resourcetype [^\.]*Site"
[command.timer: 33.946 sec]
Completed: execution time: 33.946 sec
```

atg

```
$ ctl -p DukesBank -t Updater -o staging -c Deploy
Start: "Run the coordinated deployment cycle across the configured
Sites." commands: dispatchCmd
begin workflow command (1/1) -> "dispatchCmd -command Deploy
-resourcename .* -resourcetype [^\.]*Site" ...
dispatching command: "Deploy" to: (Site) stagingDatabaseServer, (Site)
stagingApplicationServer ...
.
.
.
Completed: execution time: 15.775 sec
dispatched command: Deploy completed for: (Site) stagingDatabaseServer,
(Site) stagingApplicationServer
end workflow command (1/1) -> "dispatchCmd -command Deploy -resourcename
.* -resourcetype [^\.]*Site"
[command.timer: 15.775 sec]
Completed: execution time: 15.775 sec
```

The initial development and staging JBoss instances are available from the application server box(es) at <http://localhost:8180> and <http://localhost:8280>, or similar (depending on your node setup). You'll find the JBoss "boot.log" and "server.log" under the "default" server instance in the JBoss installation directory for each environment. e.g:

```
$ cd ~/dukesbank/development/jboss-4.0.3SP1/server/default/log
ls *.log
boot.log  server.log
```

Note:

The JBossServer type uses the "spawn" attribute of the [Exec](#) Ant task. As a result, the console output of the JBoss "run.sh" is always lost.

To recap, execution of these deployment commands completes configuration of:

- The build box hosting both the CruiseControl continuous integration service and the ControlTier AntBuilder and MavenBuilder deployments.
- The development application server box hosting JBoss.
- The development database server box hosting HSQLDB.
- The staging application server box hosting JBoss.
- The staging database server box hosting HSQLDB.

Along with the server box hosting the ControlTier server (Workbench, WebDAV & Jobcenter) you may have deployed software to as many as six boxes (or as few as one).

Note:

By this stage you will have seen what may appear to be several redundant installations of the same software be executed (Java,

ATG, JBoss). These are necessary in order to ensure each of the application "service" instance deployments can operate completely independantly of one another.

4.1.2. Execute the application and data build and deployment via Jobcenter

In [Configure Step #3](#), a new job will have been defined for the BuildAndDeploy process. After logging into Jobcenter a list of the defined jobs will be displayed. Choosing and running a job will execute the underlying BuildAndDeploy command.

The general steps to using Jobcenter to operate code and data BuildAndDeploy and Update are listed:

1. Login to Jobcenter
2. Identify the desired job
3. Run the job
4. Customize a report

Considering the specific set of jobs loaded to Jobcenter as part of the Duke's Bank demonstration:

- Locate and select the "development.BuildAndDeploy" job and use the "Choose Options and Run Job ..." button to set the "buildstamp" option and "Run Job Now" to run the application code build and deployment process against the development environment:

Note:

The "buildstamp" is equivalent to the "release version" of the application. You can choose whatever scheme appeals to you to ensure that each build is uniquely identified.

- You can follow the job's progress using the "Tail Output" view in Jobcenter
- Once the job has succeeded there will be two packages (JBossEar and HsqldbRdbDmp) in the package repository, and the development environment will running this new build and database dump.
- Similarly, using the same buildstamp, find the "developmentCatalog.BuildAndDeploy" job and follow the same process to acquire a database dump and reload it to the development environment.
- Lastly, find the "staging.Update" job and follow the same process to update both the staging application server and database instances with the selected buildstamp.

At this stage the development environment is up and running the Duke's Bank sample application: <http://localhost:8180/bank/main>, and the staging version of the application is at <http://localhost:8280/bank/main>, or similar (depending on your node setup). You can login to either site using the username "200" and the password "j2ee".

4.1.3. Execute via 'ctl'

An alternative to executing these commands via Jobcenter is to execute them directly via the `ctl` shell command. The general usage is shown below:

```
$ ctl -p project -t Updater -o name -c BuildAndDeploy -- \
    -buildstamp buildstamp
```

A typical convention is to use the date and time as the `-buildstamp` argument. For example:

```
$ ctl -p project -t Updater -o name -c BuildAndDeploy -- \
    -buildstamp 200711071500
```

If you are an experienced AntDepo user, you may also know how to run individual parts of the build and update workflow by running the appropriate command from one of the subordinate commands. For example, to run just the Deploy:

```
$ ctl -p project -t Updater -o name -c Deploy
```

Or to run just the build, you can execute the Build workflow separately:

```
$ ctl -p project -t Updater -o name -c Build -- \
    -buildstamp buildstamp
```

5. Reference

5.1. Reference

Overview

This section is useful to developers interested how the library works and users interested in knowing all the command syntax offered by the modules in this library.

This library builds on the standard ControlTier "process building blocks" - Package, Builder, Updater, Site and Deployment - each a separate workflow that compose into the single end-to-end workflow command, `BuildAndDeploy`.

General Command Workflow

The sequence diagram below describes the workflow structure defined by the modules in the library.

Type Model

The diagram below describes the inheritance and composition hierarchies defined by the core types in the library.

All the attributes and commands are defined in the Type Reference section of the

documentation. There you will find a document page for each type.

[Next: Type Reference #](#)

5.2. Types

5.2.1. Type Reference

- [ActiveMQ](#): An ActiveMQ deployment
 - [ActiveMQJMXPort](#): ActiveMQ JMX port
 - [ActiveMQOpenwirePort](#): ActiveMQ Openwire port
 - [ActiveMQRmiServerPort](#): ActiveMQ RMI Server port
 - [ActiveMQSetting](#): ActiveMQ configuration setting
 - [ActiveMQSSLPort](#): ActiveMQ SSL port
 - [ActiveMQStompPort](#): ActiveMQ Stomp port
 - [ActiveMQTimeout](#): Shutdown timeout period in seconds
- [AntBuilder](#): A simple Builder to interface with Ant
 - [BuilderAntHome](#): Ant instance that should be used
 - [BuilderJavaHome](#): Java instance that ant should use
 - [BuilderPath](#): Path that should be used
- [Apache](#): Manages an Apache HTTP server
 - [ApacheDocumentRoot](#): Directory from which httpd will serve files
 - [ApacheHttpPort](#): Apache listening port
 - [ApacheLoadModuleJk](#): Used by the LoadModule directive to link in the JK module. Filename is either an absolute path or relative to ServerRoot.
 - [ApacheLoadModulePhp](#): Used by the LoadModule directive to link in the PHP module. Filename is either an absolute path or relative to ServerRoot.
 - [ApacheServerAdmin](#): Email address of the web server administrator
 - [ApacheServerName](#): The name or address the server uses to identify itself
 - [ApacheSetting](#): An Apache configuration setting
- [CruiseControl](#): Service module for CruiseControl
 - [CruiseControlInterval](#): Schedule interval (in seconds) to be used for all projects
 - [CruiseControlJavaHome](#): Java installation to use to run Cruise Control
 - [CruiseControlJMXPort](#): JMX port used by CruiseControl
 - [CruiseControlPath](#): PATH to use to run Cruise Control
 - [CruiseControlRMIPort](#): RMI port used by CruiseControl
 - [CruiseControlSetting](#): CruiseControl configuration setting
 - [CruiseControlWebPort](#): Web port used by CruiseControl
- [ElementsProjectBuilder](#): Builds and manages projects that use the Elements modules

library

- [ElementsProjectBuilderDefaults](#): file containing project defaults properties
- [ElementsProjectBuilderSetting](#): A ElementsProjectBuilder setting.
- [ElementsProjectBuilderTemplateDir](#): file containing project template files
- [HsqldbRdb](#): A Hypersonic SQL database service
 - [HsqldbRdbHome](#): HSQLDB_HOME to use to run a Hypersonic SQL database server instance
 - [HsqldbRdbJavaHome](#): JAVA_HOME to use to run a Hypersonic SQL database server instance
 - [HsqldbRdbSetting](#): A Hypersonic SQL database configuration setting
- [HsqldbRdbDmp](#): Hypersonic SQL database dump package
- [HsqldbRdbExportBuilder](#): Hypersonic SQL database export builder
- [HsqldbRdbSchema](#): A Hypersonic SQL database schema
- [JBossEar](#): ATG Java enterprise application archive package
- [JBossServer](#): An ATG JBoss application server
 - [AppTimezone](#): A appserver timezone
 - [JBBindAddr](#): JBoss Bind Address
 - [JBConfiguration](#): The name of an ATG JBoss Configuration
 - [JBInpPort](#): The name of an ATG JBoss Configuration
 - [JBJrmpRmiObjectPort](#): JBoss JRMP RMI Object Port
 - [JBNamingRmiPort](#): JBoss Naming RMI Object Port
 - [JBossDocumentRoot](#): JBoss Document Root
 - [JBossExtractEar](#): JAVA_HOME for JBoss
 - [JBossJavaHome](#): JAVA_HOME for JBoss
 - [JBossJavaOpts](#): JAVA_OPTS for JBoss
 - [JBossPortConfig](#): JBoss service bindings port configuration
 - [JBossSetting](#): A JBoss Setting
 - [JBPooledServerBindPort](#): JBoss Pooled Server Bind Port
 - [JBTCajpPort](#): Embedded Tomcat/JBoss AJP Port
 - [JBTChttpPort](#): JBoss HTTP Port
 - [JBUIL2ServerBindPort](#): JBoss UIL2 ServerBind Port
 - [JBWSPort](#): JBoss WebServices Port
 - [JVMMaxHeap](#): JVM maximum heap size
 - [JVMMaxPermSize](#): JVM Max Perm Size memory setting
 - [JVMMinHeap](#): JVM minimum heap size
 - [JVMPPermSize](#): JVM Perm Size memory setting
- [JBossZip](#): A JBoss Distribution Package
- [MavenBuilder](#): A simple Builder to interface with Maven
 - [BuilderMavenHome](#): Maven instance that should be used

- [BuilderMavenOpts](#): Maven options that should be used
- [BuilderScmCheckExternal](#): Subversion specific external definition to check for last changed revision
- [MysqlRdb](#): A MySQL database instance
 - [MysqlBasedir](#): The MySQL installation directory
 - [MysqlDatadir](#): The MySQL data directory
 - [MysqlLoadScript](#): Schema load script used by MySQL
 - [MysqlNode](#): The node hosting the MySQL instance
 - [MysqlPassword](#): A MySQL schema username and password
 - [MysqlPort](#): Port used by MySQL
 - [MysqlSetting](#): A MySQL configuration setting
- [MysqlSchema](#): A mysql schema
- [OpenLDAP](#): OpenLDAP Lightweight Directory Access Protocol Server
 - [OpenLDAPSetting](#): OpenLDAP configuration setting
 - [OpenLDAPTimeout](#): Startup and shutdown timeout period in seconds
 - [OpenLDAPUrlList](#): Startup and shutdown timeout period in seconds
- [PlatformZip](#): Standard platform zip format package
- [Rdb](#): A relational database service
 - [RdbConnection](#): Specifies the Database Connection
 - [RdbDriver](#): Specifies the Database Driver
 - [RdbInstanceName](#): Stores the value of the database instance name
 - [RdbPassword](#): A schema user's password
 - [RdbPort](#): Specifies the Database Connection Port
 - [RdbSetting](#): an Rdb setting
 - [RDbType](#): Specifies the unique key value identifying which type database type is in use (e.g. "mssql" or "oracle", etc)
 - [RdbUserName](#): A user for a schema
- [RdbSchema](#): Represents a database schema
 - [RdbDataSourceName](#): JNDI data source name for connecting to the schema
 - [RdbMaxPoolSize](#): DataSource maximum connections
 - [RdbMinPoolSize](#): DataSource minimum connections
 - [RdbSchemaName](#): A schema name
 - [RdbSchemaPassword](#): A schema user's password
 - [RdbSchemaSetting](#): an RdbSchema setting
 - [RdbSchemaType](#): A schema name
 - [RdbSchemaUserName](#): A user for a schema
- [Tomcat](#): Mediates administration of a Tomcat server and a set of one or more associated contexts

- [TomcatContext](#): Shopzilla Tomcat web application service (extended to support a custom context and an "external" property file)
 - [TomcatContextFile](#): Shopzilla Tomcat application context path file name
 - [TomcatSetting](#): Tomcat configuration setting
- [TomcatServer](#): A tomcat deployment
 - [FilePath](#): A file path/directory
 - [JavaHomePath](#): The Java install root
 - [TomcatAjpPort](#): Tomcat Apache Jakarta Protocol port
 - [TomcatJavaOptions](#): Java runtime options (JAVA_OPTS) used when the Tomcat "start", "stop", or "run" command is executed
 - [TomcatPort](#): Port used by Tomcat
 - [TomcatProperty](#): Shopzilla Tomcat configuration property setting
 - [TomcatPropertyFile](#): Shopzilla Tomcat application configuration property file name
 - [TomcatRelease](#): Tomcat major release (e.g. 4.1, 5.x or 6.x)
 - [TomcatSecureServerPort](#): Tomcat secure server port
 - [TomcatServerPort](#): Tomcat server port
 - [TomcatSetting](#): Tomcat configuration setting
 - [TomcatShutdownPort](#): Tomcat shutdown port
 - [TomcatTimeout](#): Shutdown timeout period in seconds
- [WarUpdater](#): Updater for J2EE web archive based applications

6. All