

MyArch Solutions

Autobuild User Guide

Version 2.1 for Vangent/EFAST2 Project

4/20/2009

Contents

Autobuild Configuration	3
Installing Autobuild	3
Installation Prerequisites.....	3
Autobuild Installation Steps	4
Key Concepts	4
Autobuild Command Line	4
Key Autobuild Build Targets	5
Build Artifacts and Temporary Files	6
Environment Configuration	6
Local Environment.....	7
Dependency Management	7
Handling of Inter-Project Dependencies	7
Support for WESB Mediations.....	8
Building and Deploying WESB Mediations	8
Building WESB Mediations from Project Interchange Files (PIF)	8
Updating Imports/Exports Bindings	8
Updating Promoted SCA Properties	10
Support for WebSphere Application Server	10

Autobuild Configuration

Autobuild configuration files reside in two directories:

- `abconfig` - This directory contains settings that are not environment-specific, such as location of WESB binaries.
- `envconfig` - This directory contains environment-specific properties, such as host and port of deployment manager for different environments.

Autobuild's code is located in `autobuild-<version number>` directory. This directory should not have any project-specific settings.

"`abconfig`" must contain the file "`ab.config.properties`". This file defines global configuration settings. There could also be an optional file "`ab.local.config.properties`" with configuration settings local to the host where Autobuild is installed.

In addition to properties defined in `abconfig` and `envconfig`, any project could have a property file "`ab.build.properties`" (stored in the project's root directory). This property file specifies project-specific properties.

Autobuild loads its configuration property files in the following order:

- Project-specific `ab.build.properties` (if exists).
- "`ab.local.config.properties`" from "`abconfig`" (if exists).
- "`ab.config.properties`" from "`abconfig`".
- Environment configuration files from "`envconfig`", as described in the section "Environment Configuration".

A property that is loaded first always takes precedence. E.g., a property defined in "`ab.local.config.properties`" will take precedence over the property with the same name defined in "`ab.config.properties`".

It is recommended to always use Unix forward slash notation when specifying paths in property files or on the command line.

Installing Autobuild

Installation Prerequisites

- Apache Ant 1.7 or later.
- For building WESB mediations, WESB 6.2 must be installed on the machine. WID is not required.

- IBM JDK 1.5. If WESB is installed, you can use the JDK that comes with this product. Otherwise, you need to copy a JDK from any WAS 6.1 installation. Note that JDK 6 that comes with WID 6.2/RAD7.5 is not currently supported.

Full WAS installation is not required to be able to use Autobuild.

To install Ant, follow these steps:

- Download the Apache Ant zip from Apache Ant website and unzip it.
- Navigate to Control Panel -> System -> Advanced -> Environment Variables -> System Variables and set ANT_HOME variable to point to the directory that Ant was unzipped to, e.g., c:\devtools\apache-ant-1.7.1. Also, make sure that you have JAVA_HOME defined and that it points to IBM's JDK
- Navigate to Control Panel -> System -> Advanced -> Environment Variables -> System Variables; edit the "Path" variable to add %ANT_HOME%\bin.
- Verify the Ant installation by running "ant -version" from the command line (from any folder). You should see version 1.7.0 or later.

Autobuild Installation Steps

- Checkout autobuild-2.1, abconfig, envconfig and was-thin-admin-client-6.1 folders from \Development\utilities in Version Manager
- Define environment variable AUTOBUILD_CONFIG_HOME and point it to the location of abconfig folder.
- Add autobuild-<version>\bin to the Path environment variable.
- Navigate to abconfig and copy sample_ab.local.config.properties to ab.local.config.properties. Since every host configuration could be somewhat different, "ab.local.config.properties" file itself is not stored in VM.
- Edit serviceDeploy.install.dir to point it to WESB installation on your machine.
- Check the value of JAVA_HOME environment variable on your machine. If it does not exist or if it points to a Sun JDK, add "ibm.jdk.home" property to ab.local.config.properties and point it to the IBM JDK that is part of WESB installation, e.g.,
ibm.jdk.home=/wid62/runtimes/bi_v62/java
- To verify installation, run "ab about" from command line (from any directory).
- If Autobuild is being installed on a developer's workstation, configure local deployment environment as described later in the document.

Key Concepts

Autobuild Command Line

Autobuild runs on top of Apache Ant, so the format of Autobuild's commands is the same with Ant. Autobuild comes with a batch file "ab.bat" which is used to bootstrap the build.

Therefore, the format of an Autobuild command is "ab <target> <one or more properties>". As with Ant, properties are passed using "-D" notation, e.g.,

```
ab clean build -Denv=dev.was
```

Properties provided on the command line always override properties defined in property files.

As with Ant, you can run verbose mode using "-v" option which is useful for troubleshooting.

Key Autobuild Build Targets

Autobuild provides following build-related targets:

- **compile**. Compiles Java code.
- **test**. Runs "local" unit tests, i.e., the ones not requiring deployment to a container.
- **package**. Creates deployable unit (jar, war, ear or zip).
- **deploy**. Deploys a deployable unit to an application server. Depends on "package".
- **test.deployed**. Run soapUI or WebTest Canoo tests against a deployed application.
- **build**. Run all of the above targets.

Not all targets are available for all projects. For example, WESB mediation projects do not support "compile".

Full list of targets is available from "ab help" command.

Build targets might also be defined differently for different project/deployable unit types. For example, "package" target for a Web application will produce a "war" file whereas the same target for a POJO project will produce a jar.

Autobuild determines how to build a project based on the project's artifacts. For example, a presence of WEB-INF/web.xml file would indicate that this is a Web application, therefore Autobuild will use its web app building component. The result of this analysis is stored in the generated build file (build.xml) which is available from "build_gen" directory under the project's root.

Various administrative targets, such as WAS-related targets described later in this document, do not require project analysis and so they are executed right away.

By default, Autobuild runs the build in the current directory which is supposed be the project's root folder. This directory must contain Eclipse .project and .classpath files.

It is also possible to explicitly specify the directory using project.dir property, e.g.,

```
ab build -Dproject.dir=HelloWorld/HelloWorldWeb
```

Build Artifacts and Temporary Files

There are many files that get created during the build process. These files include class files, jar, war and ear files, testing reports.

Autobuild saves all these artifacts in the "build" directory. By default, this directory is created in the current directory. The only exception is the build of WESB/WPS SCA modules (e.g., mediations) which uses a non-standard directory for temporary and build files. The directory's location is "../ab_wpsbuild/<project name>". In other words, it is a level above the project's directory. This is done in order for WID not to pick up temporary build files. WID automatically includes all files from the project folder into the project.

It is recommended to periodically delete build directory. This is done using "clean" target.

Environment Configuration

To be able to deploy an application into an environment, Autobuild needs to know some information about this environment. For example, host and port of the deployment manager are two parameters that have to be specified for every environment.

Groups of properties pertaining to the same environment are referred to using an environment name, such as "dev.was" or "dev.esb". The environment name can be passed to Autobuild on the command line using "env" property, e.g., "ab build -Denv=dev.was"

In EFAST's case each environment correspond to a WAS cell, hence logical "dev" environment is split into "dev.was" and "dev.esb" physical environments.

The environment parameters are defined using regular Ant properties. The properties can be specified in one or more property files. The property files must reside in "envconfig" directory.

Autobuild loads environment properties at the beginning of the build process, right after the application's ab.build.properties file was loaded.

The list of property files for each environment is specified using a special property in the format "<env name>.env.filelist". This is a simple comma-delimited list. ".properties" can be omitted from file names. All paths are relative to the "envconfig" directory.

For example, "dev.esb" environment is defined as following:

```
dev.esb.env.filelist=dev.esb.env
```

These special properties must be defined in "env.metadata.properties" file also stored in "envconfig".

Currently each EAFST environment is configured using exactly one file.

Environment configuration files contain mostly WAS-related properties. Key properties include `wasadmin.host`, `wasadmin.port`, `wasadmin.user`, `wasadmin.password`. For the full list of properties, please refer to "`wasadmin_help.html`" file available from "`doc`" directory in autobuild installation.

Local Environment

Local environment refers to WESB or WAS installation on a developer's workstation. Properties for local environment are specified in the `local.env.properties`. Since potentially each installation could have different ports and user name/passwords, this file is not checked in. Instead, there is a property file call "`sample_local.env.properties`". Developers have to re-name this file into "`local.env.properties`" and make necessary changes.

To make "local" environment the default, add "`env=local`" property to `ab.local.config.properties`. Otherwise `-Denv=local` has to be provided on the command line

Dependency Management

Handling of Inter-Project Dependencies

Very often a project depends on another project for compilation and packaging. For example, a WESB mediation could depend on an SCA library. A Web application could require a build of a utility project that has to be packaged as a jar under `WEB-INF/lib`. These dependencies are specified in `.project` file. In Eclipse they are configured using "Project References". In WID they are specified using "Dependencies" screen.

Autobuild extracts this information from `.project` file and attempts to find the referenced projects. Autobuild does not use project location information from Eclipse workspace.

To find a referenced project, Autobuild first checks if there is a `<project name>.project.dir` Ant property defined, e.g., `Util.project.dir`. This property could be defined in `ab.config.property`. The value of this property is used as the project location.

If the property is undefined, Autobuild will search for the referenced project on the filesystem the level above the current project. It will look for the directories containing `.project` file with the right name.

If the project is not found, Autobuild will raise an exception.

Support for WESB Mediations

Building and Deploying WESB Mediations

You can build and deploy WESB mediations either directly from the source or from project interchange files.

To build/deploy a WESB mediation from the source, navigate to the project directory and run the following command:

```
ab <target> -Denv=<target environment>
```

Following are the available targets:

- **package** - Run WESB serviceDeploy command to create deployable EAR file
- **deploy** - Deploy EAR file to the target environment. Includes running "package". Note that the
- **test.deployed** - Run soapUI tests against previously deployed application. soapUI files must reside in <project root>/soapuitest
- **build** - Run package, deploy, test.deployed.
- **clean** - delete all temporary build files.
- **update.sca** - Update import bindings and promoted properties based on the properties defined in the environment file.

For example, to run build and deployment for DEV, run this command:

```
ab clean deploy -Denv=dev.esb
```

Building WESB Mediations from Project Interchange Files (PIF)

Targets for building WESB mediations from PIFs are the same with the ones described in the previous section. You need to provide the location of the PIF file using "project.zip" property. If this property is set, Autobuild will unzip the PIF file as well as other PIFs located in the same directory and then run the build. Autobuild needs to unzip all files (not just the one specified by "project.zip") in order to be able to find inter-project dependencies.

Example:

```
\efast\Development\ESB\MediationPIF>ab package -Dproject.zip=IFILEFilingExportImportMediation
```

Updating Imports/Exports Bindings

Autobuild is capable of changing configuration parameters of SCA exports/imports either as part of the application deployment or using "update.sca" target. The most obvious usage for this function is to change import endpoint for Web services bindings depending on the target environment. Non-web services binding types are supported as well. For example, you can use this feature to update JMS import's destination.

To understand exactly what parameters can be updated, please refer to IBM WESB/WPS InfoCenter:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v6r2mx/index.jsp?topic=/com.ibm.websphere.wps.620.doc/doc/gref_commands.html

There are several 'modify<...>Binding' commands listed on this page. Click on the link for the command for the specific binding type, e.g., modifySCAImportWSBinding or modifySCAImportHttpBinding. You can update any parameter listed under "Parameters" on the page, except for "moduleName", "import", "export", "applicationName". E.g., "endpoint" is the parameter that can be updated for Web services imports.

The syntax for specifying the parameter's value is the following:

```
<AppName>.<ModuleName>.sca.binding.<BindingName>.<ParameterName>=<new value>
```

Where:

- AppName - name of the application containing SCA modules. By default, the name is created by appending "App" to the WID's project name, e.g., StockQuoteApp. You can find out the exact name from WAS admin console.
- ModuleName - name of the SCA module. It always coincides with the project name in WID.
- BindingName - name of import or export as defined in WID. This is the name that appears on the assembly diagram.
- ParameterName - name of the parameter that will be updated, e.g., "endpoint"

Note that this syntax is independent of the binding type. Autobuild will determine the binding type based on the provided application name, module name and binding name.

Example:

```
StockQuoteApp.StockQuote.sca.binding.RealtimeService.endpoint=http://myhost/bar
```

For endpoint changes, Autobuild supports special parameter "baseURL". When this parameter is specified, Autobuild will update only the "base" portion of the URL, i.e., protocol://host:port. The resulting endpoint will be created by appending the path of the existing endpoint to the value of baseURL parameter.

For example, if an endpoint is currently set to http://myhost:8080/bar/baz and you supplied "http://newhost:8090" using "baseURL", Autobuild will set the endpoint to "http://newhost:8090/bar/baz".

Here is an example of a "baseURL" property:

```
StockQuoteApp.StockQuote.sca.binding.RealtimeService.baseURL=http://newhost:8090
```

The advantage of using "baseUrl" is that you don't need to specify fully-qualified URL for each import, only host and port.

Note that during application deployment Autobuild updates only the bindings of the application being deployed.

"sca.update" target, however, will update bindings according to the defined properties irrespective of the application. If a binding is not found, it will raise an error.

Updating Promoted SCA Properties

SCA properties can be updated much the same way with import/export bindings.

To update a SCA property, define an Autobuild property using the following format:

```
<AppName>.<ModuleName>.sca.property.<PropertyName>=<value>
```

"PropertyName" is the name of the promoted SCA property.

During application deployment Autobuild updates only the SCA properties of the application being deployed.

"sca.update" command, however, will use all promoted SCA properties irrespective of the application. If a property is not found, it will raise an error.

Support for WebSphere Application Server

Autobuild provides several deployment and WAS administration targets:

- **deploy.to.was** - Deploys EAR or WAR file or a group of files. This target is different from "deploy" target explained earlier as it does not depend on building an application.
- **bounce.app** - Restart a currently deployed application.
- **check.servers** - Check "health" of application servers in a cell
- **bounce.servers** - Restart application servers in a cell

Each target supports a number of configuration properties. For example, "deploy" allows to specify a classloader order for an application using "was.classloader.mode". A property can be supplied on a command line or specified in a property file, such as "ab.build.properties".

For example:

```
ab deploy.to.was -Dxar.file=HelloWorldWeb.war -Dwas.classloader.mode=PARENT_LAST
```

For a complete list of properties and their descriptions, please refer to "wasadmin_help.html" file available from "doc" directory in autobuild installation.

You can also use "wasadmin.help" target to obtain the same information dynamically, for example:

```
ab help.wasadmin -Dhelp.topic=check.servers
```