Automation with Apache ANT

prerequisites..

- Sound knowledge of Java programming
- Java build process
- XML
- Java web applications
- Java application packaging formats jar,war etc.
- Sound knowledge of protocols http, ftp.

Agenda

- ANT introduction
- ANT configuration in xml
- ANT documentation
- ANT in build automation
- ANT tasks
- Build targets
- ANT options and syntax
- Custom tasks with ANT
- Manage multiple build configurations
- Hands on exercises

Building the Java applications…

What are the tasks involved in building complex and large java applications ?

- ✓Define Source java files
- ✓Define configuration files xml, properties etc.
- ✓Compile the source files
- ✓Test/Debug the java classes
- ✓Package them in library/archive formats
- ✓Deploy the output files

Build Automatically ?

- Can the taska compile,test,run,package and build be done automatically ?
- Batch files support some execution tasks automatically..
- But the tasks are limited and batch files are not OS independent.
- Tedious to write the batch files.
- Reusability in batch files is limited and it needs modifications and customizations.
- Is there any simpler way to automate the build process and make it reusable across the projects and OS platforms ?.

Ant is the solution...

- Ant is the software tool program developed by Apache software foundation.
- Ant is open source and freely available.
- Ant works from command line
- Ant is built on java and used for building java projects automatically.
- Ant supports standard tasks like comile,run,copy delete etc.
- Ant also supports Custom User defined Tasks.
- Ant process further can be customized.

What is Ant?

- Java-based Build tool from Apache
- Java standard tool for building, packaging, and installing Java applications
- ANT contains a set of java classes to execute certain tasks such as create a file, compile java classes, display output etc.
- These classes are used by ant to carry out the tasks specified from the command line by the user.
- Ant Script configuration Files are written in XML

Ant Features

- Unlike make-files, Ant files work cross platform

  - No need for multiple, complex make-files depending on the operating system.

  - Tasks declared in platform independent way; Ant engine translates to OS specific commands.
- ANT is independent of File system
- Easy to create own Ant “tasks”, in addition to core tasks.

Configure the ANT

- Download the Ant distribution zip from apache web site.
- Extract the zip on your drive. It will create a Ant-Directory named 'ant-<version>'
- Add the 'Ant-Directory/bin' path to system/user environment path variable.
- This bin directory contains batch files that executes the ant classes to automate the build process.
- Add the 'JAVA_HOME' variable as your jdk directory path in system/user environment variables.i.e JAVA_HOME= C:\Program Files\Java\jdk1.5.0_14.

How Ant works ?

- Ant commands are configured in 'ant.bat' file in ant/bin directory.
- These commands/classes look for ant configuration commands in build.xml file for the current project.
- The build process execution tasks are defined in build.xml for ant to execute them as a set of targets.
- Ant loads and runs the target tasks as per the sequences specified in build.xml.
- If required you can change the name of build.xml file but then it will have to be specified as file name to ant from command line.

Ant documentation

- Navigate to manual directory inside ant installation and run the index.html in browser.
- This page contains links to all ant related tasks and information.

Test Ant

- Type “ant” at the command line from any dir
- The ant tool looks for ‘build.xml’ file in current directory to run
- Or enter “ant –file *buildfile.xml*” to specify another build file to run.
- The current directory should have the xml script file.

How to Configure build ?

- The build tasks are defined as execution targets in standard xml format in build.xml.
- Within the targets, invoke the ant tasks.
- Ant invokes the targets and runs them step by step.
- Standard Target looks in xml as shown..

```
<target name="compile"><!—Custom task name-->
  <!--Execute the 'javac' task of ANT -->
    <javac srcdir="source"
          destdir="bin"/>
  </target>
```

To execute this target run ‘ant compile’

- This configures the ANT to execute the compile target by invoking the javac command from command line.
- Number of such targets with multiple different tasks are to be defined in xml format.

Build Script XML structure

- Project declaration with Default target
- Import/include other build configuration files
- Properties/variables values definitions
- Path id declarations
- Custom tasks classes declarations
- Targets declaration
- Targets dependencies

Ant Tasks

- Each target contains one or more tasks
- Each task is a piece of executable code (e.g. javac, jar, copy, makeDir etc) supported by ant
- Tasks do the actual “build” work in Ant
- Ant has core (built in) tasks and the ability to create our own tasks

How to run targets ?

- Configure the project directory with build.xml to contain targets
- Open the command line window and change over to your project directory path.
- Run the command 'ant <targetName>'.
- For example *'ant compile'*
- Ant will look into the build.xml in current project directory and execute the specified target/s.
- If specified as default target in build.xml,no need to specify the target name with ant..simply enter 'ant' from command line.

Running multiple targets

- To run multiple targets as one after other netr them in sequence from command line..
- Ant compile run test package
- Or specify the target dependencies in xml configuration.
- <target name='package' depends ='compile,run,test'>
- So the targets compile, run and test will be executed in sequence before package target.

Task Dependencies

- You can instruct Ant to run certain tasks by specifying the dependencies so that the depended tasks will be executed by Ant first before the depending tasks.

-
```
<target name="run" depends="compileJava">
  <java fork="true" classname="Welcome">

        <!--The location of class file to be executed-->

        <classpath path="bin"/>
    </java>
  </target>
```
- Here the compileJava target task will be executed before run target.
- This enables the Ant to run the tasks in sequence and that makes the execution process Automatic!

Ant targets dependencies

- Target is a build module in Ant
- Each target contains task(s) for Ant to do
- One can be defined as a project default
- Overall structure of targets:

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>

<target name="D" depends="C,B,A"/>
```

Simple Tasks


```
<target name="prepare" depends="init“ >
   <mkdir dir="${users}" />
  </target>

  <target name="build" depends="copy" >
   <javac srcdir="src" destdir="${target}">
    <include name="**/*.java" />
   </javac>
  </target>
```

Simple Target

- <target name="clean">
- <!--Delete the output dir-->
- <delete dir="${baseDir}/bin"/>
- <delete dir="${baseDir}/config"/>
- <delete dir="${src}/out"/>
- </target>
- Enter ‘ant clean’ from command line to delete the specified directories

Ant Compile Target

- <target name="compileJava">
- <!--Execute the 'javac' task by specifying the required classpath -->
- <javac srcdir="${mySrc}"
- destdir="${baseDir}/bin"/>
- </target>

Ant run target

- <target name="run" depends="compileJava“>

  <java fork="true" classname="Welcome“>
  <!--The location of class file to run→

  <classpath path="bin"/>
  </java>
- </target>

Ant Copy task


```
<target name="copyData“>
        <copy file="Message.txt" todir="bin"/>
        <copy file="data.xml" tofile="Users.xml"/>
        <!--basedir is the current directory-->
        <copy todir="${targetDir}">
            <fileset dir="${mySrc}">
                <include name="**/*.txt"/>
                <exclude name="**/*.java"/>
                <exclude name="bin/"/>
            </fileset>
        </copy>
    </target>
```

ANT built-in Tasks

- Ant supports lots of standard built-in tasks.
- Some of the tasks supported are
  - Javac : compile java files
  - Java : Run java class
  - Copy : copy file
  - Jar : build a output jar file
  - War : build a output war file
  - copyDir : Copy the directory
  - Mkdir : create a new directory in current directory path
  - Delete : Delete the file/dir
  - Echo : Display message on console
  - Move : Move the file to new location
  - Sql : run the sql task on the database
  - Jar : Make the jar file
  - War : make the web application archive file
  - Zip : make the zip file

Default Target?

- You can specify the Ant to execute the default target without specifying it from command line.
- Just type ant from your directory path.
- This is how the default target is configured in the first line of build.xml..
- <project name="ant1" default="compileJava">
- What if you do not have default target and run just ant without specifying target ?—Nothing will happen.

    --Ant will run but no targets executed.

Ant command line options

- -help prints the options available
- -version -print the version information and exit.
- -p : prints the list of available targets in current build xml.
- -debug targetName : runs the target in debug mode.
- -lib libPath : specify additional jar libraries required to run the tasks.
- -file custonConfig.xml : specify targsts from file other than build.xml.
- -logFile : specify log file to write the log top.

Some of ant file tasks

- **Copy** Copies a file or Fileset to a new file or directory.
- **Delete** Deletes either a single file, all files and sub-directories in a specified directory,
- **Mkdir** Creates a directory
- **Move** Moves a file to a new file or directory, or a set(s) of file(s) to a new directory.
- **Concat** Concatenates multiple files into a single one or to Ant's logging system.

Interaction tasks

- **Echo** Echoes text to System.out or to a file.
- **Input** During the build process display a message and read a line of input from the console.
- **Splash** Displays a splash screen in the beginning.
- **Sql** Executes a series of SQL statements via JDBC to a database
- **Fail** Exits the current build by throwing a BuildException, optionally printing additional information.

Ant built-in properties

- **basedir** : the absolute path of the project's basedir as set with the basedir attribute of <*project*>.
- **ant.file**: the absolute path of the buildfile.
- **ant.version** :the version of Ant
- **ant.project.name** : the name of the project that is currently executing; it is set in the name attribute of <project>.
- **ant.project.default-target** : the name of the currently executing project's default target.
- **ant.project.invoked-targets** : a comma separated list of the targets that have been specified on the command line ,when invoking the current project.
- **ant.java.version** : the JVM version Ant detected.
- **ant.core.lib** : the absolute path of the ant.jar file.

Variables in build process

- In build configuration script you can define variables having values of certain path,text values or some other values for later reuse by using property tag element as
- <property name="src" location="source" />
- Refer to this variable value as ${src}.
- You can also load multiple property values defined in external file by using single statement as
- <property file="build.properties" />
- The variables defined in build.properties file are loaded and vaulable by applying the same syntax as above.

Re-use of path

- By declaring common locations under single id as path you can refer to them by using this path id.
- <path id="**compile.classpath**">
- <fileset dir="${lib.dir}">
- <include name="*.jar"/>
- </fileset>
- </path>
- Refer to path as shown here
- <target name="compile" depends="init" >
- <javac destdir="${build.classes.dir}" srcdir="src">
- **<classpath refid="compile.classpath"/>**
- </javac>
- </target>

Running java task as forked

- Java task is used to execute a Java class.
- By default this java class is run within the current running (Apache Ant) JVM,so if the java program throws exception at runtime, it terminates and also will terminate the ant build process.
- To run the java class in separate JVM proces use fork attribute with value as true,in that case the ant build process continues even if java program throw errors.

Fork and Spawn values

- fork : if enabled triggers the class execution in another JVM.
- spawn : if enabled allows to start a process which will outlive ant process and it needs fork to set as true, and is disabled by default

Control the java process

- More java task options
- classpath : the classpath to use.
- classpathref : the classpath to use, given as reference to a PATH id defined earlier.
- <jvmarg> instead elements,specify the arguments to pass to the forked JVM**.**
- maxmemory : Max amount of memory to allocate to the forked VM.
- failonerror : Stop the buildprocess if the command exits with a returncode other than 0. Default is "false"
- error :The file to which the standard error of the command should be redirected.

Passing the command line arguments at runtime

- The command line arguments are specified as nested <arg> and <jvmarg> eelements to be passed to running java class.
- <arg value="User"/>-single command-line argument with a value.
- <arg value="${cmd1}"/> pass the value of cmd1 attribute specified from command line.
- To pass multiple argument values
- <java classname="CommandClient"
  - fork="yes" failonerror="true">
  - <arg value="${cmd1}"/>
  - <arg value="${cmd2}"/>
  - <arg value="Hello"/>

Control the build versions

- Jar task while packaging allows to define the versions and custom parameters in manifest of output file.
- **BuildNumber:** task that used to track build numbers.
- <buildnumber/> Read, increment, and write a build number to the default file, build.number.
- <buildnumber file="build.dat"/>

Build date and time

- **Tstamp: s**ets the DSTAMP, TSTAMP, and TODAY properties in the current project. By default, the DSTAMP property is in the format "yyyyMMdd", TSTAMP is in the format "hhmm", and TODAY is in the format "MMMM dd yyyy". Appy the nested <format> element to specify a different format.

Manifest with jar


```
<jar destfile="${output}/Project.jar" basedir="${output}"
includes="*.class" >
        <manifest>
        <attribute name="Author" value="${user.name}"/>
        <section name="Shared">
            <attribute name="Title" value="My_Project"/>
            <attribute name="Vendor" value="Mega Ant Inc"/>
            <attribute name="Build" value="${build.number}"/>
            <attribute name="Date" value="${TODAY}"/>
        </section>
        <section name="Copyright">
            <attribute name="Copy" value="(C) Mega Ant Inc 2002"/>
        </section>
    </manifest>
```

Conditional execution of task

- The <target> element and various tasks such as <fail> and task elements support if and unless attributes which can be used to control whether the item is run or otherwise takes effect.
- <target name="Check-Property" if="User">
- <echo message="The 'User' property is present"/>
- </target>

Unless attribute

- The unless attribute disables the condition if it is either the name of property which is defined, or if it evaluates to a true-like value.
- <target name="Check_PropAbsence" unless="Work">
- <echo message=“The 'Work' property is not defined"/>

More condition checks

- **Available** : Sets a property if a specified file, directory, class in the classpath, or JVM system resource is available at runtime.
  - If the resource is present, the property value is set to true by default; otherwise, the property is not set. You can set the value to something other than the default by specifying the value attribute.
- **Condition** : Sets a property if a certain condition holds true.
  - The conditions to test are specified as nested elements
  - Conditions are specified as nested elements, you must specify exactly one condition.
  - If the condition holds true, the property value is set to true by default, otherwise, the property is not set. You can set the value to other than the default by specifying the value attribute.

Nested conditions

- **Available**: test if the resource/property is available
- **Os:T**est whether the current operating system is of a given type.
- **Equals:** Tests whether the two given values are equal.
- **Isset:** Test whether a given property has been set in this project.
- **http:** checks for a valid response from a web server of the specified url.
- **Contains:** Tests whether a string contains specified value.
- **Istrue:** Tests whether a value equals true
- **Isfalse:** Tests whether a value is not true.
- **Matches :** Test if the specified string matches the specified regular expression pattern.
- **Resourcecount:** used to test the size of a resource collection. A single Resource Collection should be specified via a nested element or the refid attribute.

Combining the conditions

- **Not**: expects exactly one other condition to be nested into this element, negating the result of the condition. It doesn't have any attributes and accepts all nested elements of the condition task as nested elements as well.
- **And**: This condition is true if all of its contained conditions are true, conditions will be evaluated in the order they have been specified in the build file.
- **Or**: accepts an arbitrary number of conditions as nested elements. This condition is true if at least one of its contained conditions is true, conditions will be evaluated in the order they have been specified in the build file.
- **Xor**: performs an exclusive or on all nested elements and evaluate the result as true or false.

Running the tasks in parallel

- By default all the tasks are executed in sequence, but certain task you can specify to run them in parallel with each other by using parallel task.
- The parallel task executes nested tasks in parallel with no guarantees of thread safety.
- Every task will run in its own thread.
- Used to take advantage of available processing resources to execute external programs simultaneously.
- Use to test servers, where the server can be run in one thread and the test harness is run in another thread.

Parallel task attributes

- threadCount : Maximum numbers of thread to use.
- threadsPerProcessor :Maximum number of threads to use per available processor
- timeout : Number of milliseconds before execution is terminated.
- failonany : If any of the nested tasks fails, execution of the task completes at that point without waiting for any other tasks to complete.

Sequential

- The parallel task may be combined with the *sequential* task to define sequences of tasks to be executed on each thread within the parallel block.
- The sequential task can contain other ant tasks. The nested tasks are simply executed in sequence.
- Sequential's primary use is to support the sequential execution of a subset of tasks within the parallel task
- Any valid Ant task may be embedded within the sequential task.

Define your own task

- Ant has predefined number of tasks which have mapping to ant library java classes responsible for executing certain tasks.
- You can define your own task by defining a java class and mapping the class to certain task name.
- For custom atsk..
- Define a java class by extending ant class Task and override the execute method.
- Map the java class to custom xml task element in build file with taskdef element.
- Use the custom xml task name within the target.

Custom Task

User specific custom tasks can be defined by extending the Ant 'Task' class and overriding the execute method.

```
public class CustomTask extends Task {

    public void execute() {
       log("Running Custom Task !",
           Project.MSG_INFO);
        System.out.println("Custom Task running");
    }

}
```

Configure custom task in xml

- The custom task is specified in build.xml
- <taskdef name="MyCustomTask"
classname="com.user.CustomTask"
classpath="MyTask.jar"/>
- This custom task is invoked from target element
- <target name="runCustomTask">
    < MyCustomTask />
- </target>
- Parameters can be passed to custom tasks as well..

Customizing tasks

- The custom tasks can be defined to accept parameters as attributes.
- The custom task elements can have nested child elements.
- The custom task elements can produce certain results as well.

Custom Task Class structure

- The class must extend org.apache.tools.ant.Task
- The should have a no-arg constructor.(default)
- The class should override execute method.
- In case of attribute support the class should have variable defined by same name and corresponding set method defined to set the value.
- To support text content values, define an addText method
- public void addText(String text)

Custom Structure 2

- For each nested child element, define a create or add method
  - To add empty child task elements the method is public *ChildTask createChildTask() where* ***ChildTask must be the name of a*** class that also follows these steps.
  - For non-empty child task elements define a method as public void add*ChildTask(ChildTask child)*
- Define/override the method that implements the tasks
  - public void execute()

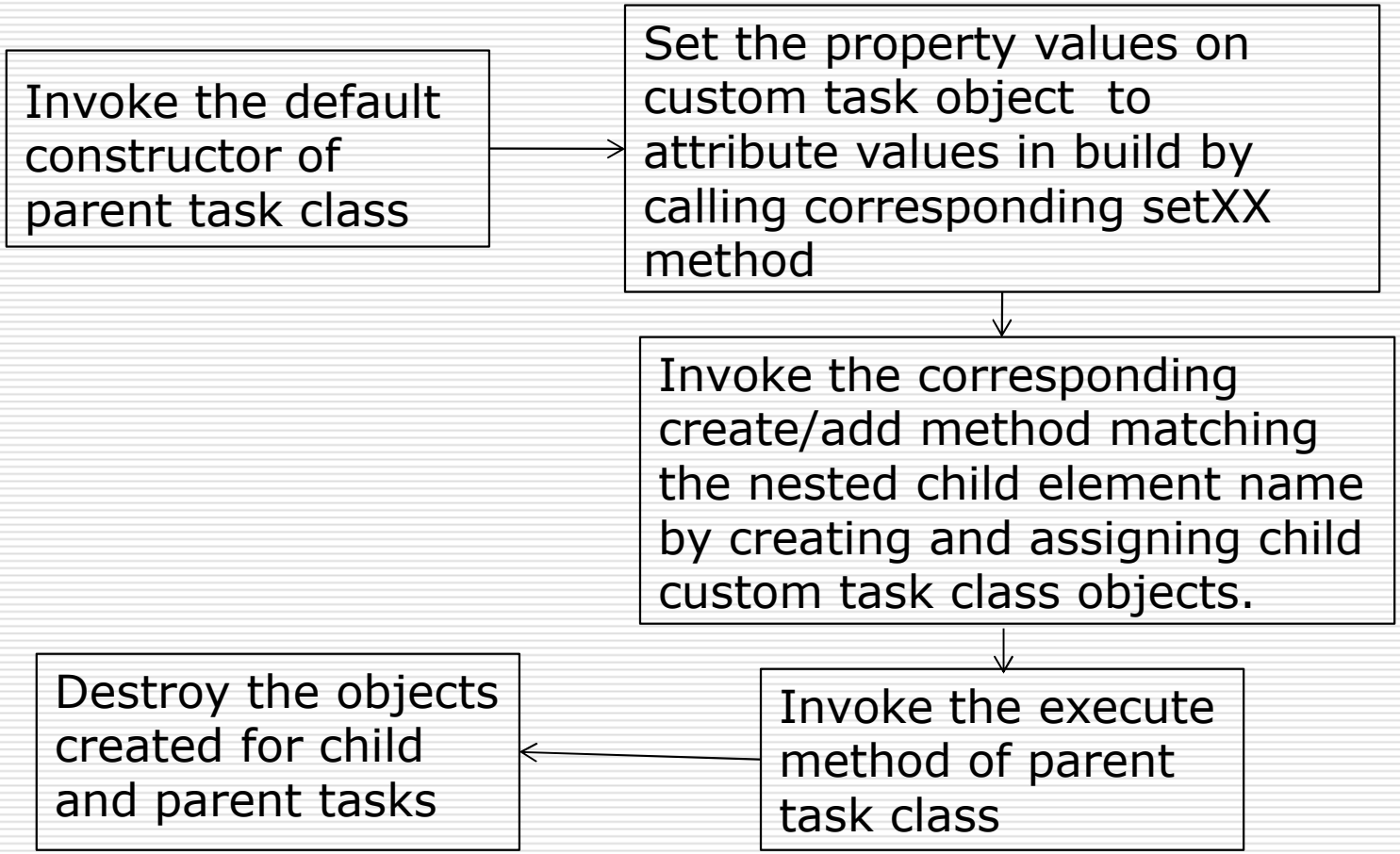
Using the custom task

- Compile the custom task class
- package it in jar or ensure that it can be found using the CLASSPATH environment variable.
- Define user defined task name mapped to custom task class name by using teskdef element.
- Invoke the user defined task name within a target.

Custom task usage

- <taskdef name="**custom2**" classname="com.user.CustomTask" **classpath**="MyTask.jar"/>
- <path id="libPath">
- <fileset dir="${My_lib}">
- <include name="MyTask.jar"/>
- </fileset>
- </path>
- <target name="runCustomTask">
- **<custom2/>**
- </target>

Custom Task Life cycle in Ant framework



Tomcat server custom Tasks

- The Tomcat web server library supports following custom tasks
- Deploy: Deploy the war file into running server.
- List : Lists the number of applications currently deployed.
- Reload : Reload a specific war/web application.
- Resources : Execute resource task for a application
- Roles: Define user and roles for server
- Start: Start the Tomcat web server.
- Stop : Stop the running Tomcat server
- UnDeploy : Undeploy an application currently running in the Tomcat web server.

Other custom tasks

- Ant supports following additional custom tasks available in ant installation
- Junit Task: To run the Junit test cases
- TestReport : To generate the test report in html/xml format
- JavaDoc : To generate java documentation in html format from the java source files.
- Sql: To run sql commands on database sever.
- Remote Execution tasks

Re-use the xml code

- Another way to support the customized behavior is to be able to call the the predefined block mof xml code in the current build.
- This is supported by defining xml block of code in macrodef tag definitions.
- The macrodef is alternative to defining java custom task class and customizing the build process.

MacroDef task

- It defines a new task using a <sequential> nested task as a template.
- Nested elements <attribute> and <element> are used to specify attributes and elements of the new task.
- These get substituted into the <sequential> task when the new task is run.

Macrodef task attributes

- Attribute nested element is used to specify attributes of the new task in macrodef tag.
- The values of the attributes get substituted into the templated task.
- The attributes will be required attributes unless a default value has been set.
- <macrodef name=" text_attrib_macro ">
- ***<attribute name="message" default="This is a common message"/>***

Macrodef task elements

- **Element** is used as nested tag in macrodef tag to specify nested elements of the new task.
- The contents of the nested elements of the task instance are placed in the templated task at the tag name.
- The case of the element name is ignored.
- <macrodef name="child-contents-macro">
- **<element name="content"/>**

Using the macrodef definitions

- <target name=“Usage" description=“Use those macros here">
- <text_attrib_macro="My Message to all the users of ant,beware! it also bites.."/>
- < text_attrib_macro /><!—default message-->
- < child-contents-macro >
- Use this way for longer text routes...
- </ child-contents-macro >
- <target

Execute non-java/native processes

- **Exec** task is to execute a system command.
- If the *os* attribute is specified, then the command is only executed when Apache Ant is run on one of the specified operating systems.
- The nested <arg> elements are used to specify additional argument values to run the process e.g. to run notepad and open data.txt file use as shown...
- <exec executable="notepad.exe”>
- <arg value="data.txt"/>
- </exec>

Run the child process in separate thread

- By default the process started is executed in ant own process and control returns to ant process only when the child process returns or terminates.Till that ant process remains blocked.
- To avoid this use spawn attribute to specify child process execution in separate thread other than ant so ant will not get blocked.
- <exec executable="notepad.exe" **spawn='true**'>
- <arg value="data.txt"/>
- </exec>

child process parameters

- nested <arg> elements are used to specify command line arguments to be passed to child process.
- To specify the environment variables to be passed to the child process use nested <env> elements with key –value pairs.
- timeout : Stop the command if the child process doesn't finish within the specified time (given in milliseconds).
- failonerror : Stop the build-process if the command exits with a return code signaling failure. Defaults to false.

Wait for the child process

- The <waitfor> task is used to block execution until a set of specified conditions become true. This is intended to be used with the *parallel* task to synchronize a set of processes.
- The conditions to wait for are defined in nested elements, if multiple conditions are specified, then the task will wait until all conditions are true..
- The <sleep> task is used to specify a fizxed delay.

Wait for example

- <exec executable="D:\\apache-tomcat-6.0.24\\bin\\startup.bat" spawn="false">
  </exec>
- <waitfor maxwait="25" maxwaitunit="second" checkevery="2000" timeoutproperty="30">
- <!--check the url for every 5000 miliseconds if not found active-->
- <http url="http://localhost:8082"/>
- </waitfor>

Waitfor returns

- If both maxwait and maxwaitunit are not specified, the maxwait is 3 minutes (180000 milliseconds).
- timeoutproperty : the name of the property to set if maxwait has been exceeded.
- If the timeoutproperty attribute has been set, a property of the specified name will be created if the condition didn't come true within the specified time.
- You can use nested conditions within <waitfor> that satisfy the certain results/conditions.

Integrate multiple targets

- AntCall task : Call another target within the same buildfile optionally specifying some properties (passing parameters).
- **This task must not be used outside of a target.**
- By default, all of the properties of the current project will be available in the new project.
- Setting the *inheritAll* attribute to false and only "user" properties (i.e., those passed on the command-line) will be passed to the new project.
- The set of properties passed to the new project will override the properties that are set in the new project

Calling the target from another build...

- ***Ant*** : Runs Apache Ant task on a supplied another buildfile from within a current target.
- This can be used to build subprojects. **This task must not be used outside of a target if it invokes the same build file it is part of.**
- When the *antfile* attribute is omitted, the file "build.xml" in the supplied directory (*dir* attribute) is used.
- If no target attribute is supplied, the default target of the new project is used.

Properties to child build from parent build

- By default, all of the properties of the current project will be available in the new project.
- By setting the *inheritAll* attribute to false and only "user" properties (i.e., those passed on the command-line) will be passed to the new project.
- The set of properties passed to the new project will override the properties that are set in the new project.

Use multiple build configurations

- Import and include specifies importing or including the targets from other build files into current build configuration.

Import targets

- Imports another build file into the current project.
- It works like as if the imported file was contained in the importing file, minus the top <project> tag.
- The import task may only be used as a top-level task. This means that it may not be used inside a target.
  - target overriding
  - special properties

Target overriding in import

- If a target in the main file is also present in at least one of the imported files, the one from the main file takes precedence.
- So if import a *docsbuild.xml* file named **builddocs**, that contains a "**docs**" target, we can redefine it in main buildfile and that is the one that will be called.

Targets for the imported..

- This makes it easy to keep the same target name, so that the overriding target is still called by any other targets--in either the main or imported buildfile(s)--for which it is a dependency,
- With a different implementation in imported build,the target from *docsbuild.xml* is made available by the name "**builddocs.docs**". This enables the new implementation to call the old target, thus *enhancing* it with tasks called before or after it.
- If specified the *as* attribute of the task, its value will be used to prefix the overriden target's name instead of the name attribute of the project tag.

Properties for child imported build...

- Imported files are treated as they are present in the main buildfile.
- Ant adds a property that contains the path to the imported buildfile. With this path, the imported buildfile can keep resources and be able to reference them relative to its position.
- So if I import for example a *docsbuild.xml* file named **builddocs**, I can get its path as **ant.file.builddocs**, similarly to the **ant.file** property of the main buildfile.

Include child build xml/s

- Include another build file into the current project.
- On execution it will read another Ant file into the same Project rewriting the included target names and depends lists.
- The include task may only be used as a top-level task. This means that it should not be used in a target.

Target override in include

- Any target in the included file will be renamed to *prefix.name* where *name* is the original target's name and *prefix* is either the value of the *as* attribute or the *name* attribute of the *project* tag of the included file.
- The depends attribute of all included targets is rewritten so that all target names are prefixed as well. This makes the included file self-contained.
- Properties and reference values inheriting from the parent build is also configurable here..

Task execution on Remote machines

- To run certain commands on reamote machine from local machine ant supportts **SSHEXEC** task**.**
- For this it requires an SSH daemon running on remote machine.
- The attributes for remote machine
  - host :The hostname or IP address of the remote host to which you wish to connect.
  - username :The username on the remote host to which you are connecting.
  - command :The command to run on the remote host.
  - port The port to connect to on the remote host**.** Default is 22**.**

Remote Exec configuration

- timeout : Stop the command if it doesn't finish within the specified time (given in milliseconds Defaults to 0 which means "wait forever".
- input : a file from which the executed command's standard input is taken.
- output : The name of a file to which to write the output of remote executed process.
- outputproperty :The name of a property in which the output of the command should be stored.
- <sshexec host="myPC" username=“john" password=“bipsData123" command=“openFile"/>

Protocol tasks

- **FTP:** The ftp task implements a basic FTP client that can send, receive, list, delete files, and create directories on remote machines by applying ftp protocol.
- **Mail :** task to send SMTP email. It can send mail using either plain text, UU encoding, or MIME format mail, depending on what is available.
- **Telnet:** Task to automate a remote telnet session.

Ant role in build automation

- Ant is used in every stage of application build process.
- Ant can integrate with number of build processes internal or external to ant.
- Ant integrated with third party application servers and software suites to manage the testing,reporting,documenting, packaging processes.

Continuous Integration Suites

- The Continuous Integration suites are available which continuously nitrate the software build process by monitoring the code change,test results, periodic builds etc.
- CruiseControl,Hudson are the CI suites available which integrate with CVS servers, JUnit test frameworks and Eclipse/Netbeans build environments and trigger the build process periodically by configuring customized tasks with ant.

Thanks to you!!