ANT

- Official ANT site:
  - http://ant.apache.org
- Wikipedia topic:
  - http://en.wikipedia.org/wiki/Apache_Ant

ANT - Introduction

- Apache Ant is a Java-based build tool. In theory, it is kind of like Make, without Make's wrinkles.
- Don't worry about Make if you don't know what it is.
- http://www.gnu.org/software/make/
  - If you still want to know what make is.

ANT - Why?

- Ant is cross-platform, you can use it any of the commonly available OS like windows, Unix and mac.
- Ant can be extended by writing Java classes, again the extensions are portable across platform thanks to Java.
- Configuration files are XML based – commonly used configuration language.

ANT - Installation

- The latest stable version of Ant is available from the Ant web page http://ant.apache.org/
- Setup
  - Add the bin directory to your path.
  - Set the ANT_HOME environment variable to the directory where you installed Ant.
  - Optionally, set the JAVA_HOME environment variable. This should be set to the directory where your JDK is installed.
  -

Using ANT

- Ant's build files are written in XML.
- Each build file contains one project and at least one (default) target.
- Targets contain task elements.
- Each task element of the build file can have an id attribute and can later be referred to by the value supplied to this. The value has to be unique.

ANT buildfile

- Ant's build files are named 'build.xml' and stored in your project folder.
  - NOTE: 'build.xml' is the convention. You can use any other name you like. If a different name used, ant will not find the build by default.
- Ex:
  - /home/user/project1
  - /home/user/project1/build.xml
- ant -projecthelp [prints all targets in build.xml]

buildfile <project> element

- <project> is the root element. All build.xml begins with this element.

```
<project name="MyProject" default="dist" basedir=".">
   <description>
      simple example build file
   </description>
</project>
```

- name, default and basedir are the 3 attributes of <project> element.

<project> element attributes

- name
  - The name of the project (can be anything)
- default
  - the default target to use when no target is supplied.
- basedir
  - the base directory from which all path calculations are done. If the property is not set, the parent directory of the build file will be used.

buildfile <target> element

- Each project defines one or more targets.
- A target is a set of tasks you want to be executed.
- When starting Ant, you can select which target(s) you want to have executed.
- When no target is given, the project's default is used.
- A target can depend on other targets.

buildfile <target> element

```
<project name="MyProject" default="dist" basedir=".">
     <description>
       simple example build file
     </description>
    <target name="init">
     <!-- Create the time stamp -->
     <tstamp/>
     <!-- Create the build directory structure used by compile
    -->
     <mkdir dir="${build}"/>
    </target>
</project>
```

buildfile <property> element

- A project can have a set of properties.
- These might be set in the build file by the property task, or might be set outside Ant.
- A property has a name and a value; the name is case-sensitive.
- Properties may be used in the value of task attributes by placing the property name between "${" and "}" in the attribute value.

buildfile <property> element

```
<project name="MyProject" default="dist" basedir=".">
    <description>
        simple example build file
    </description>
          <!-- set global properties for this build -->
          <property name="src" location="src"/>
          <property name="build" location="build"/>
          <property name="dist"  location="dist"/>
          <target name="init">
            <!-- Create the time stamp -->
            <tstamp/>
            <!-- Create the build directory structure used by compile -->
            <mkdir dir="${build}"/>
          </target>
</project>
```

System Properties

- Ant provides access to all system properties as if they had been defined using a <property> task.
- System Properties such as..

Key	Description of Associated Value
java.version	Java Runtime Environment version
java.vendor	Java Runtime Environment vendor
java.vendor.url	Java vendor URL
java.home	Java installation directory

  - And many more system properties...

Built-in Properties

- In addition, Ant has some 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.java.version = the JVM version Ant detected; currently it can hold the values "1.2", "1.3", "1.4" and "1.5".

ANT Buildfile - Example

```
<project name="MyProject" default="dist" basedir=".">
  <description>
    simple example build file
  </description>
 <!-- set global properties for this build -->
 <property name="src" location="src"/>
 <property name="build" location="build"/>
 <property name="dist"  location="dist"/>

 <target name="init">
  <!-- Create the time stamp -->
  <tstamp/>
  <!-- Create the build directory structure used by compile
    -->
  <mkdir dir="${build}"/>
 </target>

 <target name="compile" depends="init"
    description="compile the source " >
  <!-- Compile the java code from ${src} into ${build} -->
  <javac srcdir="${src}" destdir="${build}"/>
 </target>
<target name="dist" depends="compile"
    description="generate the distribution" >
  <!-- Create the distribution directory -->
  <mkdir dir="${dist}/lib"/>

  <!-- Put everything in ${build} into the MyProject-
    20061213.jar file -->
  <jar jarfile="${dist}/lib/MyProject-20061213.jar" basedir="$
    {build}"/>
 </target>

 <target name="clean"
    description="clean up" >
  <!-- Delete the ${build} and ${dist} directory trees -->
  <delete dir="${build}"/>
  <delete dir="${dist}"/>
 </target>
</project>
```

Path-like structures

- You can specify PATH- and CLASSPATH-type references using both ":" and ";" as separator characters. Ant will convert the separator to the correct character of the current operating system.

```
<classpath>
  <pathelement path="${classpath}"/>
  <pathelement location="lib/helper.jar"/>
</classpath>
```

Running ANT

- running Ant from the command-line is simple: just type ant.
  - When no arguments are specified, Ant looks for a build.xml file in the current directory and, if found, uses that file as the build file and runs the target specified in the default attribute of the <project> tag.
  - To make Ant use a build file other than build.xml, use the command-line option -buildfile file, where file is the name of the build file you want to use.

Running ANT - Options

```
ant [options] [target [target2 [target3] ...]]
Options:
  -help, -h              print this message
  -projecthelp, -p       print project help
     information
  -version               print the version
     information and exit
  -debug, -d             print debugging
     information
  -lib <path>            specifies a path to search
     for jars and classes
  -logfile <file>        use given file for log
    -l      <file>                 "
  -logger <classname>    the class which is to
     perform logging
```

```
-buildfile <file>      use given buildfile
  -file    <file>                 "
  -f       <file>                 "
-D<property>=<value>   use value for given
   property
-keep-going, -k        execute all targets that
   do not depend on failed target(s)
-propertyfile <name>   load all properties
   from file with -D properties taking
   precedence
-find <file>           (s)earch for buildfile
   towards the root of the filesystem and use
   it
-noclasspath           Run ant without using
   CLASSPATH
-main <class>          override Ant's normal
   entry point
```

Tasks

- Task - a piece of code that can be executed.
  - A task can have multiple attributes. The value of an attribute might contain references to a property. These references will be resolved before the task is executed.
- Tasks have a common structure:

  <name attribute1="value1" attribute2="value2" ... />

  - where name is the name of the task, attributeN is the attribute name, and valueN is the value for this attribute.

ANT Tasks - Categorized

- Given the large number of tasks available with Ant, it may be difficult to get an overall view of what each task can do.

- Archive Tasks
- Audit/Coverage Tasks
- Compile Tasks
- Deployment Tasks
- Documentation Tasks
- EJB Tasks
- Execution Tasks
- File Tasks
- Java2 Extensions Tasks
- Logging Tasks
- Mail Tasks
- Miscellaneous Tasks
- .NET Tasks
- Pre-process Tasks
- Property Tasks
- Remote Tasks
- SCM Tasks
- Testing Tasks

ANT Tasks - Reference

- Copy
  - Copies a file or resource collection to a new file or directory. By default, files are only copied if the source file is newer than the destination file, or when the destination file does not exist. Note: overwrite attribute will explicitly overwrite files.

- Copy a single file

  <copy file="myfile.txt" tofile="mycopy.txt"/>
- Copy a single file to a directory

  <copy file="myfile.txt" todir="../some/other/dir"/>
- Copy a directory to another directory

```
<copy todir="../new/dir">
 <fileset dir="src_dir"/>
</copy>
```

ANT Tasks - Reference

- Delete
  - Deletes a single file, a specified directory and all its files and subdirectories, or a set of files specified by one or more resource collections.

- Delete a single file
  <delete file="/lib/ant.jar"/>
- Delete a directory
  <delete dir="lib"/>
- Deletes all files and subdirectories of build, including build itself

```
<delete
  includeEmptyDirs="true">
  <fileset dir="build"/>
</delete>
```

ANT Core Tasks

- Echo
- Exec
- Jar
- Java
- Javac
- Import
- Mkdir
- Property
- Sleep
- Sql
- Tar
- Touch
- Unzar
- Unzip
- War
- Zip

Tutorial – Hello World

- Preparing the Hello World project
  - We want to separate the source from the generated files, so our java source files will be in src folder. All generated files should be under build, and there split into several subdirectories for the individual steps: classes for our compiled files and jar for our own JAR-file.

Hello World – Manual Steps

- **Manual not using Ant**

Directory creation:

- md src
- md build\classes

HelloWorld.java

```
package oata;
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Compile:

```
javac -sourcepath src -d build\classes
    src\oata\HelloWorld.java
```

Run:

```
java -cp build\classes oata.HelloWorld
```

Jar:

```
echo Main-Class:
    oata.HelloWorld>myManifest
md build\jar
jar cfm build\jar\HelloWorld.jar myManifest -C
    build\classes .
java -jar build\jar\HelloWorld.jar
```

Hello World – Manual to ANT

- Manual to Ant - thinking the build process
- Possible targets that evolved from the previous example
  - Compile
  - Jar
  - Run
- Its a good practice to have a 'clean' target
  - Clean
- Lets write the build.xml in the next slides.

Hello World - build.xml

```
<project>

    <target name="clean">
        <delete dir="build"/>
    </target>
    <target name="compile">
        <mkdir dir="build/classes"/>
        <javac srcdir="src"
     destdir="build/classes"/>
    </target>
    <target name="run">
        <java jar="build/jar/HelloWorld.jar"
     fork="true"/>
    </target>

    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar
     destfile="build/jar/HelloWorld.jar"
     basedir="build/classes">
            <manifest>
                <attribute name="Main-Class"
     value="oata.HelloWorld"/>
            </manifest>
        </jar>
    </target>

</project>
```

Hello World – Invoking Build

- Now you can compile, package and run the application via
  - ant compile
  - ant jar
  - ant run
- Or shorter with
  - ant compile jar run

Manual vs ANT - Comparison

JAVA

```
md build\classes
javac
    -sourcepath src
    -d build\classes
    src\oata\HelloWorld.java
echo Main-Class: oata.HelloWorld>mf
md build\jar
jar cfm
    build\jar\HelloWorld.jar
    mf
    -C build\classes
    .

java -jar build\jar\HelloWorld.jar
```

ANT

```
<mkdir dir="build/classes"/>
<javac
    srcdir="src"
    destdir="build/classes"/>
<mkdir dir="build/jar"/>
<jar
    destfile="build/jar/HelloWorld.jar"
    basedir="build/classes">
    <manifest>
        <attribute name="Main-Class"
     value="oata.HelloWorld"/>
    </manifest>
</jar>
<java jar="build/jar/HelloWorld.jar"
     fork="true"/>
```

Hello World – Enhancing Build

- Some enhancements:
  - many time referencing the same directories
  - main-class and jar-name are hard coded
  - the right order of build steps required to run.
- The first and second point would be addressed with properties, the third with a special property - an attribute of the <project>-tag and the fourth problem can be solved using dependencies.

Hello World – build.xml (update)

```
<project name="HelloWorld" basedir="."
      default="main">
   <property name="src.dir"     value="src"/>
   <property name="build.dir"   value="build"/>
   <property name="classes.dir"
      value="build/classes"/>
 <property name="jar.dir"  value="build/jar"/>
  <property name="main-class"
      value="oata.HelloWorld"/>
   <target name="clean">
       <delete dir="build"/>
   </target>
   <target name="compile">
       <mkdir dir="${classes.dir}"/>
       <javac srcdir="src" destdir="$
     {classes.dir}"/>
   </target>
<target name="jar" depends="compile">
    <mkdir dir="${jar.dir}"/>
  <jar destfile="${jar.dir}/apache-ant.jar"
  basedir="${classes.dir}">
         <manifest>
             <attribute name="Main-Class"
  value="${main-class}"/>
         </manifest>
    </jar>
</target>
<target name="run" depends="jar">
    <java jar="${jar.dir}/apache-ant.jar"
  fork="true"/>
</target>
 <target name="clean-build"
  depends="clean,jar"/>
<target name="main" depends="clean,run"/>
</project>
```

Hello World – Invoke New Build

- Now it's easier, just do 'ant' and you will get

```
Buildfile: build.xml
clean:
compile:
    [mkdir] Created dir: C:\...\build\classes
    [javac] Compiling 1 source file to C:\...\build\classes
jar:
    [mkdir] Created dir: C:\...\build\jar
      [jar] Building jar: C:\...\build\jar\HelloWorld.jar
run:
     [java] Hello World
main:
BUILD SUCCESSFUL
```

ANT Core Types - PatternSet

- PatternSet
  - Patterns can be grouped to sets and later be referenced by their id attribute. They are defined via a patternset element, which can appear nested into a FileSet or a directory-based task that constitutes an implicit FileSet. In addition, patternsets can be defined as a stand alone element at the same level as target — i.e., as children of project as well as as children of target.

ANT Core Types - PatternSet

- PatternSet

```
<patternset id="non.test.sources">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</patternset>
```

  - Builds a set of patterns that matches all .java files that do not contain the text Test in their name. This set can be referred to via <patternset refid="non.test.sources"/>, by tasks that support this feature, or by FileSets.

ANT Core Types - Selectors

- Selectors are a mechanism whereby the files that make up a <fileset> can be selected based on criteria other than filename as provided by the <include> and <exclude> tags.
  - How to use a Selector - A selector is an element of FileSet, and appears within it. It can also be defined outside of any target by using the <selector> tag and then using it as a reference.

Types – Available Selectors

- <contains> - Select files that contain a particular text string
- <date> - Select files that have been modified either before or after a particular date and time
- <depend> - Select files that have been modified more recently than equivalent files elsewhere
- <depth> - Select files that appear so many directories down in a directory tree
- <different> - Select files that are different from those elsewhere
- <filename> - Select files whose name matches a particular pattern. Equivalent to the include and exclude elements of a patternset.
- <present> - Select files that either do or do not exist in some other location
- <containsregexp> - Select files that match a regular expression
- <size> - Select files that are larger or smaller than a particular number of bytes.
- <type> - Select files that are either regular files or directories.
- <modified> - Select files if the return value of the configured algorithm is different from that stored in a cache.
- <signedselector> - Select files if they are signed, and optionally if they have a signature of a certain name.
- <scriptselector> - Use a BSF or JSR 223 scripting language to create your own selector

Selector Examples - Contains

- Here is an example of how to use the Contains Selector:

```
<fileset dir="${doc.path}" includes="**/*.html">
   <contains text="script" casesensitive="no"/>
</fileset>
```

  – Selects all the HTML files that contain the string script.

Selector Examples - Date

- Here is an example of how to use the Date Selector:

```
<fileset dir="${jar.path}" includes="**/*.jar">
    <date datetime="01/01/2001 12:00 AM" when="before"/>
</fileset>
```

  - Selects all JAR files which were last modified before midnight January 1, 2001.

Selector Examples - Depend

- Here is an example of how to use the Depend Selector:

```
 <fileset dir="${ant.1.5}/src/main"
includes="**/*.java">
    <depend targetdir="${ant.1.4.1}/src/main"/>
 </fileset>
```

  - Selects all the Java source files which were modified in the 1.5 release.

ANT Core Types - FileSet

- FileSet
  - A FileSet is a group of files. These files can be found in a directory tree starting in a base directory and are matched by patterns taken from a number of PatternSets and Selectors.

```
<fileset dir="${server.src}" casesensitive="yes">
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</fileset>
```

  - Groups all files in directory ${server.src} that are Java source files and don't have the text Test in their name.

FileSet using PatternSet

- FileSet
  - PatternSets can be specified as nested <patternset> elements. In addition, FileSet holds an implicit PatternSet and supports the nested <include>, <includesfile>, <exclude> and <excludesfile> elements of PatternSet directly, as well as PatternSet's attributes.

```
<fileset dir="${server.src}" casesensitive="yes">
  <patternset id="non.test.sources">
    <include name="**/*.java"/>
    <exclude name="**/*Test*"/>
  </patternset>
</fileset>
```

  - Groups the same files as the above example, but also establishes a PatternSet that can be referenced in other <fileset> elements, rooted at a different directory.

FileSet using Selectors

- FileSet
  - Selectors are available as nested elements within the FileSet. If any of the selectors within the FileSet do not select the file, the file is not considered part of the FileSet. This makes a FileSet equivalent to an <and> selector container.

```
<fileset dir="${server.src}" casesensitive="yes">
  <filename name="**/*.java"/>
  <filename name="**/*Test*" negate="true"/>
</fileset>
```

  - Groups all files in directory ${client.src}, using the <filename> selector.

build.xml using build.properties

build.properties

- JAVA_HOME=/usr/java/jdk1.6.0
- PROJECT_PATH=.
- VERSION=0.73
- SOURCE_PATH=${PROJECT_PATH}/src
- BUILD_PATH=${PROJECT_PATH}/build
- DIST_PATH=${PROJECT_PATH}/dist
- CLASSES_PATH=${BUILD_PATH}/classes/production/sted
- TEST_CLASSES_PATH=${BUILD_PATH}/classes/test/sted
- DEPLOY_PATH=${BUILD_PATH}/bin
- STED.JAR_NAME=sted.jar
- STED-WIDGETS.JAR_NAME=sted-widgets.jar
- STED.JAR=${DEPLOY_PATH}/${STED.JAR_NAME}
- STED-WIDGETS.JAR=${DEPLOY_PATH}/${STED-WIDGETS.JAR_NAME}
- STED.ZIP_NAME=sted-${VERSION}.zip
- STED.ZIP=${DIST_PATH}/${STED.ZIP_NAME}
- STED-FULL.ZIP_NAME=sted-with-src-${VERSION}.zip
- STED-FULL.ZIP=${DIST_PATH}/${STED-FULL.ZIP_NAME}

- NOTE:
  - 'build.xml' uses 'build.properties' by property task

    <property file="build.properties" description="STED User Environment Settings"/>
  - build.properties co exists with build.xml in the same folder

build.xml using build.properties

```
<project name="STED" default="init" basedir=".">

   <property file="build.properties" description="STED User Environment
       Settings"/>

   <target name="init">
      <tstamp/>
      <!-- Create the build directory structure used by compile -->
      <mkdir dir="${BUILD_PATH}"/>
      <mkdir dir="${CLASSES_PATH}"/>
      <echo message="Project Path:    ${PROJECT_PATH}"/>
      <echo message="Source Path:     ${SOURCE_PATH}"/>
      <echo message="Classes Path:    ${CLASSES_PATH}"/>
      <echo message="Deploy Path:     ${DEPLOY_PATH}"/>
      <echo message="Jar Path:        ${STED.JAR}"/>
   </target>

   <path id="SOURCE_PATH">
      <pathelement location="${SOURCE_PATH}"/>
   </path>

   <path id="CLASS_PATH">
      <pathelement location="${CLASSES_PATH}"/>
   </path>
```

```
<target name="copy.resource" depends="init">
   <copy todir="${BUILD_PATH}" verbose="true">
      <fileset dir="${PROJECT_PATH}" excludes="**/build/**, **/dist/**,
    **/test/**" defaultexcludes="true"/>
   </copy>
</target>

<target name="compile" depends="init" description="compiles
    source">
   <javac srcdir="${SOURCE_PATH}" destdir="${CLASSES_PATH}"
    deprecation="on" verbose="true">
   </javac>
   <!--HACK for development builds only-->
   <!--copy 'config' folder in classess.. since ResourceBundle would
    fail-->
   <!--Deployment does not require this hack, because STED_HOME
    is set in classpath-->
   <copy todir="${CLASSES_PATH}" verbose="true">
      <fileset dir="${PROJECT_PATH}" includes="config/**"
    defaultexcludes="true"/>
   </copy>
</target>
```

build.xml using build.properties

```
<target name="deploy.widgets" depends="compile"
    description="creates sted-widgets.jar">
   <jar basedir="${CLASSES_PATH}" destfile="${STED-
    WIDGETS.JAR}" includes="**/intellibitz/sted/widgets/**">
      <manifest>
          <attribute name="Built-By" value="IntelliBitz Technologies.,
    Muthu Ramadoss."/>
          <section name="STED Widgets - ReUsable Swing
    Components.">
              <attribute name="Specification-Title" value="STED"/>
              <attribute name="Specification-Version" value="$
    {VERSION}"/>
              <attribute name="Specification-Vendor" value="IntelliBitz
    Technologies.,"/>
              <attribute name="Implementation-Title" value="STED"/>
              <attribute name="Implementation-Version" value="$
    {VERSION} ${TODAY}"/>
              <attribute name="Implementation-Vendor" value="IntelliBitz
    Technologies.,"/>
          </section>
      </manifest>
   </jar>
</target>
```

```
<target name="deploy.sted" depends="copy.resource, compile,
    deploy.widgets" description="creates sted.jar">
   <jar basedir="${CLASSES_PATH}" destfile="${STED.JAR}"
    includes="**/intellibitz/**" excludes="**/config/**,
    **/intellibitz/sted/widgets/**">
      <manifest>
          <attribute name="Main-Class" value="intellibitz.sted.Main"/>
          <attribute name="Built-By" value="IntelliBitz Technologies.,
    Muthu Ramadoss."/>
          <section name="STED - Free OpenSource
    Transliterator/Editor.">
              <attribute name="Specification-Title" value="STED"/>
              <attribute name="Specification-Version" value="$
    {VERSION}"/>
              <attribute name="Specification-Vendor" value="IntelliBitz
    Technologies.,"/>
              <attribute name="Implementation-Title" value="STED"/>
              <attribute name="Implementation-Version" value="$
    {VERSION} ${TODAY}"/>
              <attribute name="Implementation-Vendor" value="IntelliBitz
    Technologies.,"/>
          </section>
      </manifest>
   </jar>
</target>
```

build.xml using build.properties

```
<target name="clean.deploy" depends="clean.build, deploy.sted"
     description="Cleans and Creates Deployment"/>

<target name="undeploy" description="removes sted.jar">
    <delete verbose="true">
        <fileset dir="${DEPLOY_PATH}" includes="$
     {STED.JAR_NAME}"/>
    </delete>
</target>

<target name="clean.classes" description="deletes classes">
    <delete includeemptydirs="true" verbose="true">
        <fileset dir="${CLASSES_PATH}" excludes="**/.dependency-
     info/**" defaultexcludes="false"/>
    </delete>
</target>

<target name="clean.build" description="deletes build directory">
    <delete includeemptydirs="true" verbose="true">
        <fileset dir="${BUILD_PATH}" excludes="**/.dependency-info/**"
     defaultexcludes="false"/>
    </delete>
</target>

<target name="release.sted" depends="clean.deploy"
     description="Creates STED Production Deliverable">
    <jar basedir="${BUILD_PATH}" destfile="${STED.ZIP}"
     excludes="**/classes/**, **/src/**, **/temp/**, ${STED.ZIP}, $
     {STED-FULL.ZIP}">
    </jar>
</target>

<target name="release.sted-full" description="Creates STED
     Production Deliverable">
    <jar basedir="${BUILD_PATH}" destfile="${STED-FULL.ZIP}"
     excludes="**/classes/**, **/temp/**, ${STED.ZIP}, ${STED-
     FULL.ZIP}">
    </jar>
</target>

<target name="run" depends="deploy.sted" description="runs sted">
    <!-- WINDOWS -->
    <!--<exec dir="${DEPLOY_PATH}" executable="sted.bat"/>-->
    <!-- LINUX -->
    <exec dir="${DEPLOY_PATH}" executable="./sted.sh"/>
</target>

</project>
```

Summary

- Ant is a cross-platform build tool for Java.
- Ant uses XML based configuration file typically named 'build.xml'.
- Project, Targets and Tasks
  - A build.xml would contain 1 project with one or more targets and each of the target containing one or more tasks.
- Core and Optional tasks provided by Ant.
- Ant requires and uses Java platform.

Resources

- Official ANT site:
  - http://ant.apache.org
- ANT Manual:
  - http://ant.apache.org/manual/index.html
- Wikipedia topic:
  - http://en.wikipedia.org/wiki/Apache_Ant
- Example IntelliBitz project:
  - http://sted.svn.sourceforge.net/viewvc/sted/FontTransl

About IntelliBitz Technologies

- http://training.intellibitz.com
- http://groups.google.com/group/etoe
- http://sted.sourceforge.net
  - 168, Medavakkam Main Road, Madipakkam
  - Chennai, 600091, India.
  - +91 44 2247 5106
  - training@intellibitz.com

ANT says Good Bye!

Thank You!