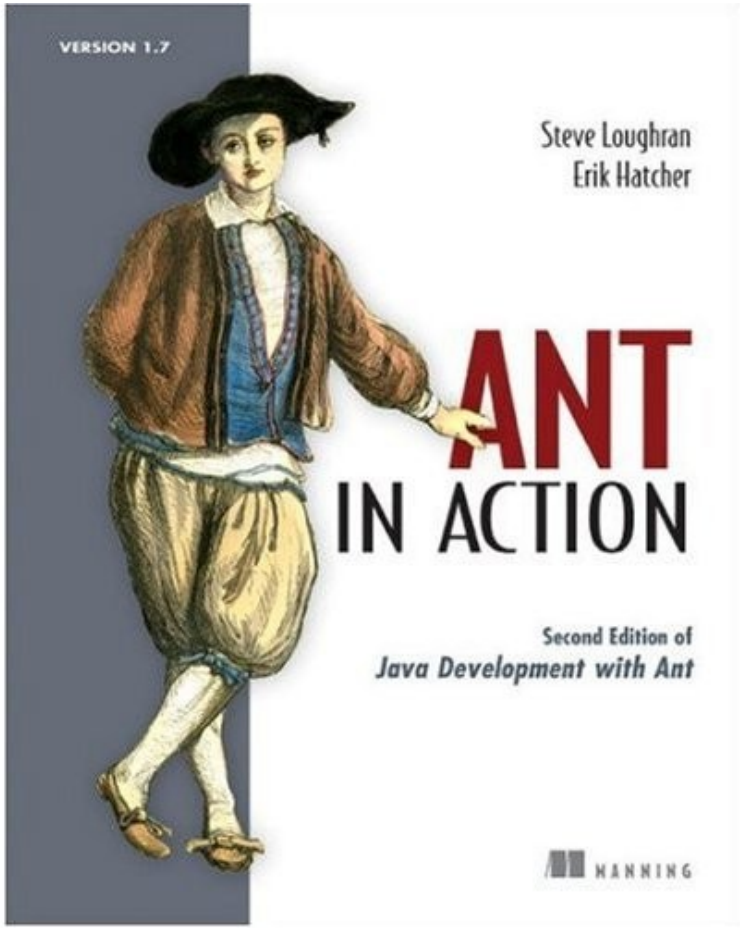
ApacheCon
leading the wave of open source

07EU
May 1-4 2007
Amsterdam, Netherlands

Extending Ant

Steve Loughran
stevel@apache.org

About the speaker



Research on deployment at HP Laboratories:
http://smartfrog.org/

Ant team member

Shipping in May/June:
Ant in Action!

Today's Topic: Extending Ant

- ✔ Inline scripting
- ✔ Ant Tasks
- ✔ Conditions
- ✔ Ant Types and Resources
- ✗ Embedded Ant
- ✗ Non-XML syntaxes
- ✗ Cutting your own Ant distribution

Out of scope for today
Ask on dev@ant.apache.org

Before we begin

Ant 1.7 + source

- Apache BSF
- jython, jruby, groovy,javascript netREXX,...

or

- Java1.6

```
> ant -diagnostics
------- Ant diagnostics report -------
Apache Ant version 1.7.0

bsf-2.3.0.jar (175348 bytes)
jruby-0.8.3.jar (1088261 bytes)
js-1.6R3.jar (708578 bytes)
jython-2.1.jar (719950 bytes)


java.vm.version : 1.6.0-b105
```

Problem

Generate a random number as part of the build

<scriptdef> declares scripted tasks

```
<scriptdef language="javascript" manager="javax"
  name="random">
  <attribute name="max"/>
  <attribute name="property"/>

var max=attributes.get("max")
var property=attributes.get("property")
if(max==null || property==null) {
  self.fail("'property' or 'max' is not set")
} else {
  var result=java.util.Random().nextInt(max)
  self.log("Generated random number " + result)
  project.setNewProperty(property, result);
}

</scriptdef>
```

Script Language

All attributes are optional

Java API

Ant API

What is in scope in <scriptdef>?

`self`	the active subclass of `ScriptDefBase`
`self.text`	any nested text
`attributes`	map of all attributes
`elements`	map of all elements
`project`	the current project

<scriptdef> tasks *are* Ant Tasks

```
<target name="testRandom">

  <random max="20" property="result"/>

  <echo>Random number is ${result}</echo>
</target>
```

```
> ant testRandom

Buildfile: build.xml

testRandom:
   [random] Generated random number 8
     [echo] Random number is 8

BUILD SUCCESSFUL
Total time: 1 second
```

Yes, but do they work?

```
<random max="20" property="result"/>
<random max="20"/>
```

No working tasks without tests!

Imagine: test targets with assertions

```
<target name="testRandomTask">
  <random max="20" property="result"/>
  <echo>Random number is ${result}</echo>
  <au:assertPropertySet name="result"/>
  <au:assertLogContains
   text="Generated random number"/>
</target>
```

```
<target name="testRandomTaskNoProperty">
  <au:expectfailure expectedMessage="not set">
    <random max="20"/>
  </au:expectfailure>
</target>
```

AntUnit

```
<target name="antunit"
    xmlns:au="antlib:org.apache.ant.antunit">
  <au:antunit>
    <fileset file="${ant.file}"/>
    <au:plainlistener/>
  </au:antunit>
</target>
```

```
>ant antunit
Buildfile: build.xml

antunit:
[au:antunit] Build File: /home/ant/script/build.xml
[au:antunit] Tests run: 3, Failures: 0, Errors: 0,
             Time elapsed: 0.057 sec
[au:antunit] Target: testRandomTask took 0.038 sec
[au:antunit] Target: testRandomTaskNoProperty took 0.018 sec

BUILD SUCCESSFUL
```

http://ant.apache.org/antlibs/antunit/

AntUnit is JUnit for Ant

- `<antunit>` can test any number of nested files
- All targets matching test?* are run
- `setup` and `teardown` targets for every test
- plain text or XML output
- Assert state of build and file system
- `<expectfailure>` probes fault handling.

```
<assertTrue>
<assertFalse>
<assertEquals>
<assertPropertySet>
<assertPropertyEquals>
<assertPropertyContains>
<assertFileExists>
<assertFileDoesntExist>
<assertDestIsUptodate>
<assertDestIsOutofdate>
<assertFilesMatch>
<assertFilesDiffer>
<assertReferenceSet>
<assertReferenceIsType>
<assertLogContains>
```

Nested Elements in <scriptdef>

```
<scriptdef language="ruby" name="nested"
    uri="http://antbook.org/script">
  <element name="classpath" type="path"/>
  paths=$elements.get("classpath")
  if paths==nil then
    $self.fail("no classpath")
  end
  for path in paths
    $self.log(path.toString())
  end
</scriptdef>
```

ant type;
use classname to give a full
Java class name

```
<target name="testNested"
  xmlns:s="http://antbook.org/script">
  <s:nested>
    <classpath path=".:${user.home}"/>
    <classpath path="${ant.file}" />
  </s:nested>
</target>
```

`<scriptdef>` best practises

- ✔ Use `<scriptdef>` first!
- ✔ Java 1.6+: target JavaScript
- ✔ Test with `<antunit>`
- ✔ Declare tasks into new namespaces

XML Namespaces

Tasks and types are declared in Ant's main namespace

- Unless you set the `uri` attribute of any -def task (`<scriptdef>`, `<typedef>`, `<presetdef>`, ...)
- Private namespaces give isolation from the rest of Ant.

```
<target name="testRandom"
    xmlns:s="http://antbook.org/script">

  <s:random max="20" property="result"/>

  <echo>Random number is ${result}</echo>
</target>
```

Ant is more flexible about naming than most XML languages
—you don't need to declare children or attributes
in the same namespace as the parent

Other scriptable things

`<script>`	Inline script (obsolete)
`<scriptfilter>`	Inline filter of native/Java I/O
`<scriptcondition>`	Scripted condition (set `self.value` to true/false)
`<scriptselector>`	File selection logic in a script
`<scriptmapper>`	Filename mapping for `<copy>`, `<uptodate>`, `<apply>`...

use in emergency, but they have limited reuse except through build file sharing

Writing a “classic” Java task

```
public class ResourceSizeTask extends Task {
  private String property;
  private Union resources = new Union();

  public void execute() {
      if (property == null) {
          throw new BuildException("No property");
      }
  }
}
```

1. extend `org.apache.tools.ant.Task`
2. override `public void execute()`
3. throw `BuildException` when things go wrong

Public setter methods become attributes

```
public void setProperty(String property){
  this.property = property;
}

public void setFile(File file)
public void setLimit(int limit)
public void setClasspath(Path path)
public void setFailonerror(boolean flag)
```

- Ant expands properties then converts the string to the required type
- Anything with a String constructor is supported
- Files and paths are resolved to absolute paths
- Overloaded methods? String comes last

Add elements through add() and create()

```
public void addSrc(FileSet fileset) {
  resources.add(fileset);
}

public Path createClasspath() {
  Path p=new Path(getProject());
  resources.add(p);
  return p;
}

public void add(ResourceCollection rc) {
  resources.add(rc);
}
```

Compile, <taskdef>, then use

```
<taskdef name="filesize"
   uri="http://antbook.org/"
   classname="org.antbook.apachecon.ResourceSizeTask"
   classpath="${build.classes.dir}"/>


<target name="testPath" xmlns:book="http://antbook.org/">
  <book:filesize property="size">
    <path path="${java.class.path}"/>
  </book:filesize>
  <au:assertPropertySet name="size"/>
</target>
```

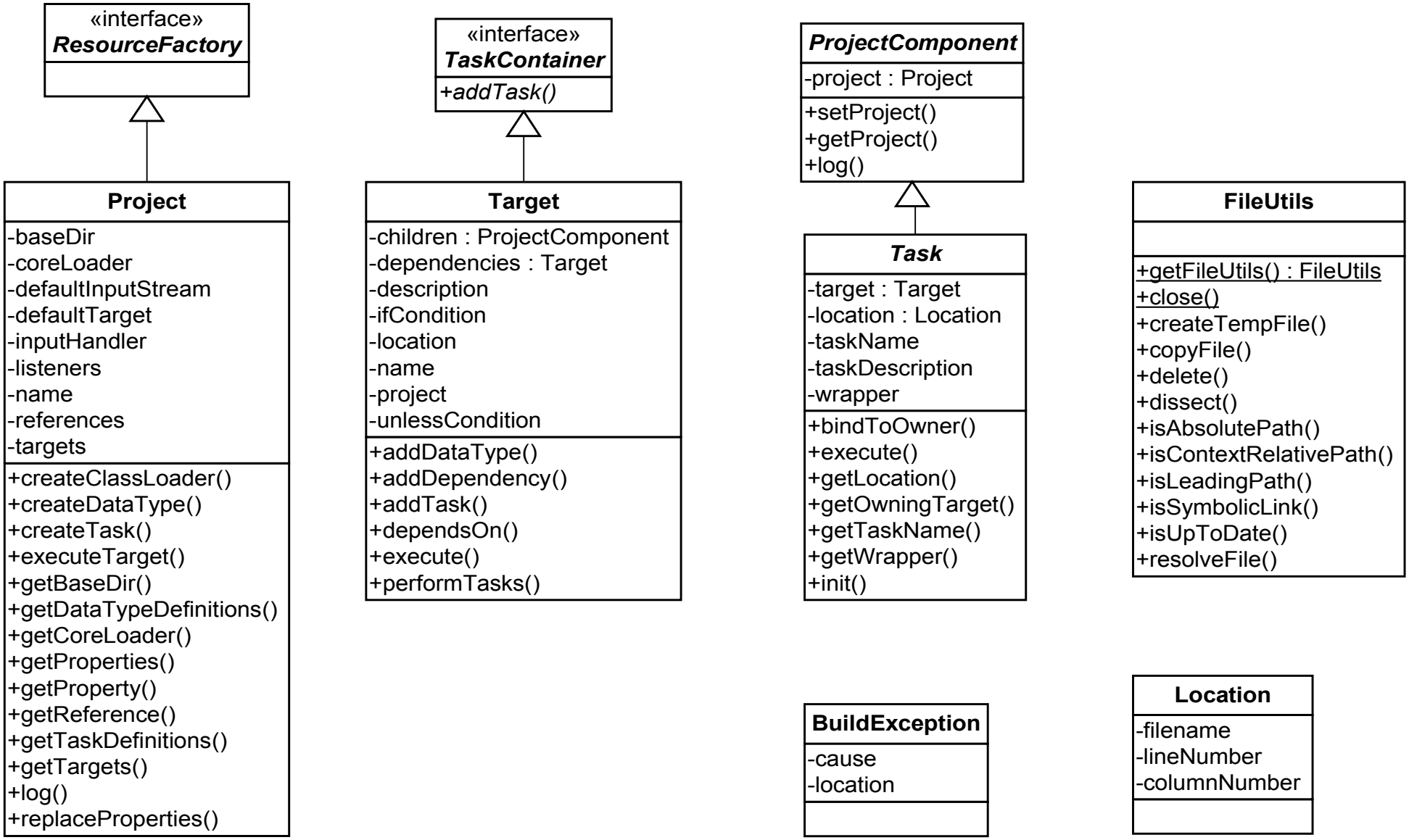
Nested Text

```
package org.antbook.apachecon;
import org.apache.tools.ant.Task;

public class MessageTask extends Task {

    private String text = "";

    public void addText(String text) {
        this.text = getProject().replaceProperties(text);
    }

    public void execute() {
        log(text);
    }
}
```

explicitly expand properties

Once you forget to expand properties (e.g. <sql>), you cannot patch it back in without running the risk breaking build files out in the field.

Ant's Main Classes

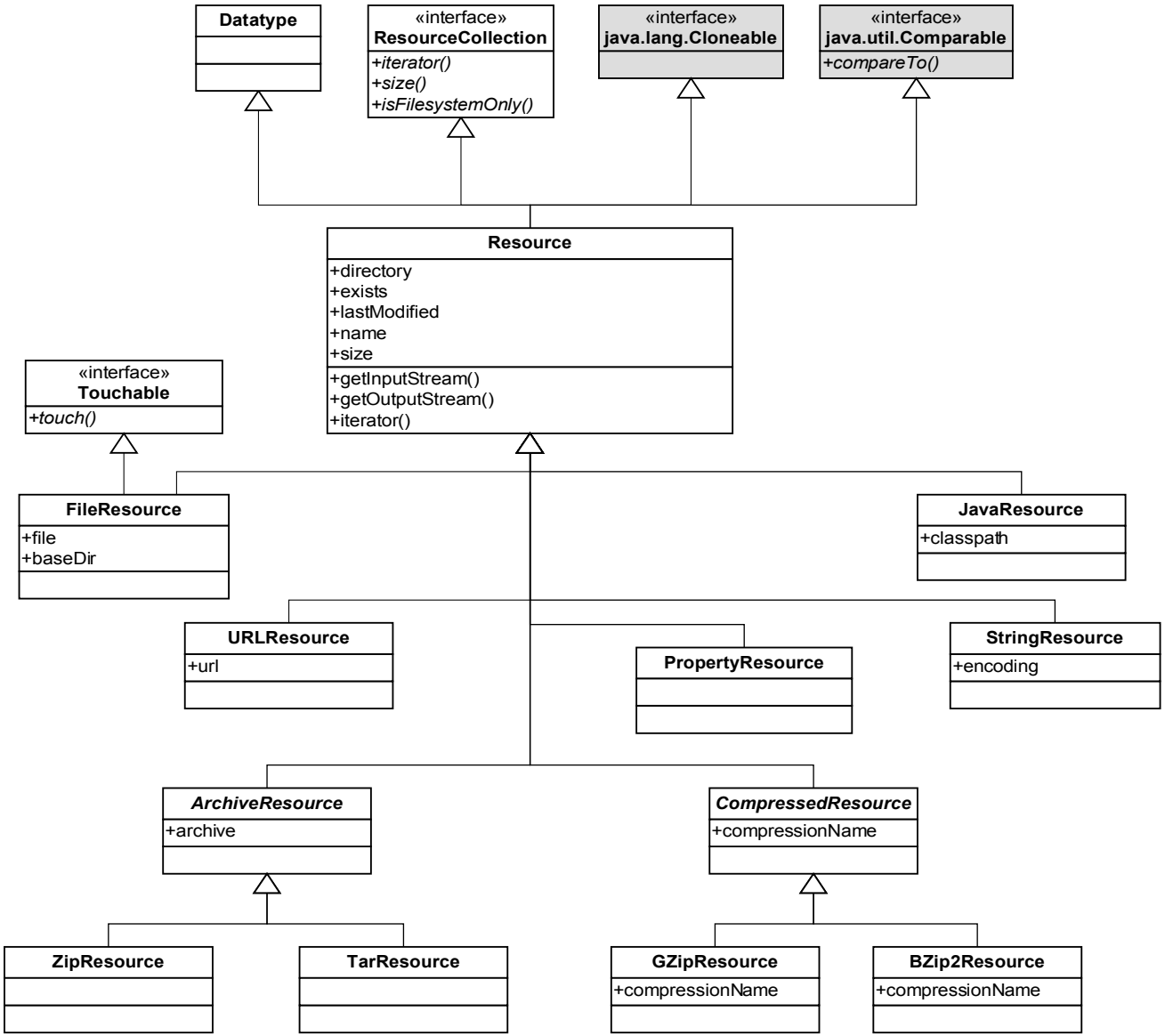


Resources

- Resources are Ant datatypes that can be declared and cross-referenced across tasks
- Resources are sources and sinks of data
- Some resources are *Touchable*
- Resources may be out of date or not found

Resources provide task authors with a way of modelling data, and of integrating with existing tasks that work with resources

Resources


Datatype
«interface»
ResourceCollection
+iterator()
+size()
+isFilesystemOnly()
«interface»
java.lang.Cloneable
«interface»
java.util.Comparable
+compareTo()
Resource
+directory
+exists
+lastModified
+name
+size
+getInputStream()
+getOutputStream()
+iterator()
«interface»
Touchable
+touch()
FileResource
+file
+baseDir
JavaResource
+classpath
URLResource
+url
PropertyResource
StringResource
+encoding
ArchiveResource
+archive
CompressedResource
+compressionName
ZipResource
TarResource
GZipResource
+compressionName
BZip2Resource
+compressionName

A simple resource

```
public class RandomResource extends Resource
  implements Cloneable {

  public boolean isExists() {
      return true;
  }

  public long getLastModified() {
      return UNKNOWN_DATETIME;
  }

  public InputStream getInputStream() throws IOException {
      if(isReference()) {
          RandomResource that;
          that = (RandomResource) getCheckedRef();
          return that.getInputStream();
      } else {
         //TODO
      }
  }
}
```

timestamp logic

reference resolution

return the data

Use <typedef> to define resources

```
<typedef name="rdata"
  uri="antlib:org.antbook.resources"
  classname="org.antbook.apachecon.RandomResource" />

<copy todir="build">
  <res:rdata name="random.bin" length="8192"/>
</copy>

<res:rdata length="10" id="sharedResource"/>

<loadresource property="random.property">
  <resource refid="sharedResource"/>
</loadresource>
<echo>random=${random.property}
```

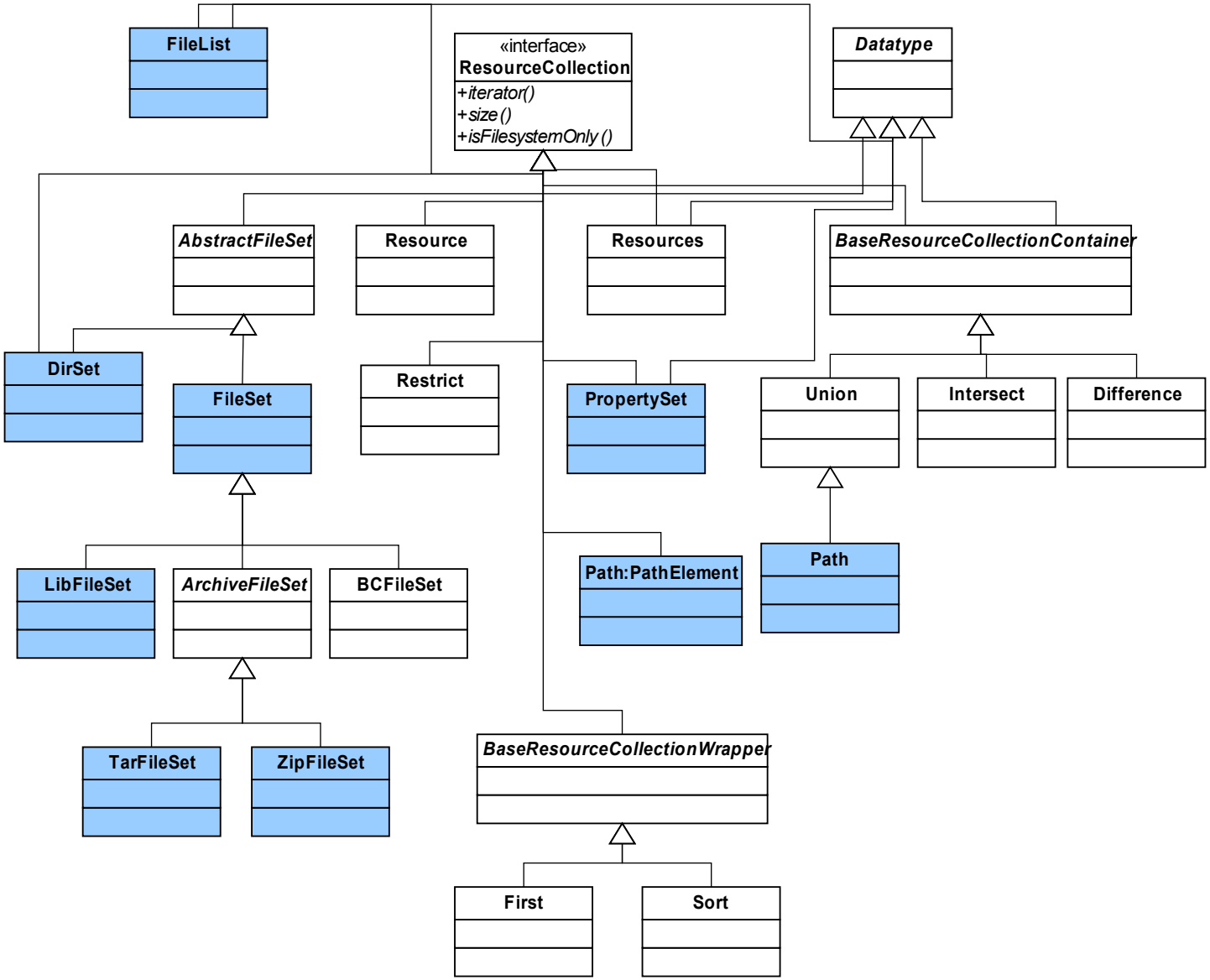
Demo

```
> ant demo
Buildfile: build.xml

demo:
     [echo]
random=yijyaeaxakikeybgbfvhbyottpqtnvpauiemymnibbancaxcolbdptvbeyuhhqj
msroanrjjsmnocyqhyoibytsugdwfsqsecsnugcijnjnndhuuodbjoiiknsutukfhwrtos
afbujkhvgaeypfagrnvvcwqtkxjicxyuxnkqikujjtmwopkemeiwsitjpuieqxpehdfvwk
drdtspbbftrjipnwvfwviooxwhfhslkfvbeywwucfykglccoakyvrmncvwhmpycsojqbnf
kogrlkutuyugklmqkoyludubsaumcpvirgtjwghsukiphippruonyekcqdklkuwlruesse
vkbffgrljeiotgohcfjuftnplvitkfcrbsmrevhlonsjojqogkrvtcrborxexxlnpkrjva
ovgqusombwyuxorlilavjkbwgjkfuxvsknmvtgxdbcddmgqufifehyfugvirofybecfrsm
ejhkxrbgwmpxkucrelggfllqchuamadseihfmuefcavmwgasdncqfejtfombgsiqhnfaig
pyfjtjuglftrjksnnvcwskdrjgqilgogvubbwghgoefivsqntdimlgmntqgghshoqgdeal
kjfpbcmoadcexraveoglcqdfdmyskngyfxtgqwlmobuvphxywkdpaeketobferskqcbtpc
xxvfvaonkiymweeosgnceynernu

BUILD SUCCESSFUL
```

Resource best practises

- ✔ Use resources to add new data sources/destinations to existing tasks
- ✔ Declare resources into new namespaces
- ✔ Test with <antunit>
- ✔ Package as an antlib

Resource Collections



Resource-enabling a task

Store in a Union

Accept any collection

iterate!

```
public class ResourceCount extends Task {
  private Union resources = new Union();

  public void add(ResourceCollection rc) {
      resources.add(rc);
  }

  public void execute() {
    int count = 0;
    Iterator element = resources.iterator();
    while (element.hasNext()) {
      Resource resource = (Resource) element.next();
      log(resource.toString(),Project.MSG_VERBOSE);
      count++;
    }
    log("count="+count);
  }
}
```

Conditions

```
public class isEven extends ProjectComponent
  implements Condition {

  protected int value;

  public void setValue(int v) {
    value = v;
  }

  public boolean eval() {
    return (value & 1) == 0;
  }

}
```

- A condition is a component that implements `Condition` and `Condition.eval()`
- Declare it with `<typedef>`

Turning a JAR into an antlib

- Add `antlib.xml` to your package/JAR
- Implicit loading via the XML namespace URL

```
xmlns:lib="antlib:org.antbook.resources"
```

```
<antlib>
  <taskdef name="filesize"
   classname="org.antbook.tasks.filesize.ResourceSizeTask"/>

  <typedef name="rdata"
   classname="org.antbook.resources.RandomResource" />
</antlib>
```

declare tasks, types, presets, macros, scriptdefs

Using an antlib

Implicit loading via the XML namespace URL
`xmlns:res="antlib:org.antbook.resources"`

```
<project name="filesize" default="default"
    xmlns:au="antlib:org.apache.ant.antunit"
    xmlns:res="antlib:org.antbook.resources">

  <res:rdata name="random.bin" size="8192"/>
</project>
```

or by explicitly loading

```
<typedef
  onerror="failall"
  uri="antlib:org.antbook.resources"
  classpath="${resources.jar}"/>
```

Summary

- *Ant is a collection of Java classes that are designed to be extended*
- Extend Ant through script, Java tasks, resources
- Ant: conditions, selectors, filters, new antlib-types.
- Embrace AntUnit!
- See the Ant source, Ant in Action, and the Ant mailing list for more details.

Vragen?

Embedded Ant

✗ Don't call Ant `Main.main(String args[])` directly

✓ Create a new Project and execute it

```
Project project = new Project();
project.init();
DefaultLogger logger = new DefaultLogger();
project.addBuildListener(logger);
logger.setOutputPrintStream(System.out);
logger.setErrorPrintStream(System.err);
logger.setMessageOutputLevel(Project.MSG_INFO);
System.setOut(
  new PrintStream(new DemuxOutputStream(project, false)));
System.setErr(
  new PrintStream(new DemuxOutputStream(project, true)));
project.fireBuildStarted();
```

Cutting your own Ant distribution

- ✗ Don’t break existing Ant installations
- ✗ Don’t be incompatible with normal Ant releases
- ✗ Don’t hide the Ant classes in a different JAR.
- ✓ Build with all the optional libraries present
- ✓ Rename ant.bat and ant.sh
- ✓ Create custom equivalents of ANT_HOME, ANT_OPTS, ANT_ARGS env variables.
- ✓ Document what you've done

WebLogic and WebSphere: please help us here!