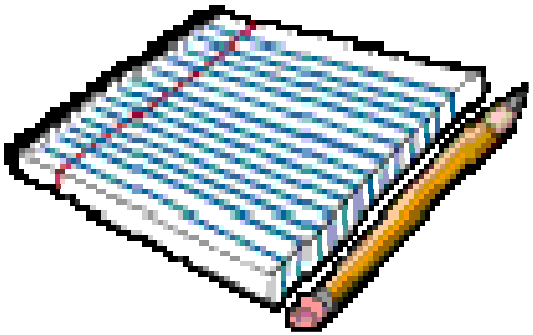
Rapid application development with Apache Turbine and Maven

Henning Schmiedehausen

<henning@apache.org>



The Apache Jakarta Project

http://jakarta.apache.org/

Introduction

- This tutorial shows how to develop web applications with the Turbine web framework
- Apache Maven is used as the build tool which glues the build process together.
- This tutorial will show the basics
- Target is to get you „over the hump“

About the Speaker

- Working with Turbine since 2001 (Turbine 2.1)
- Committer on the Turbine project (Fall 2002)
- ASF member (May 2005)
- Did many of the reworkings in the 2.2 -> 2.3 development cycle and made the 2.3 and 2.3.1 releases
- Developed and deployed ~ 20 Turbine based applications ranging from 1-500 kLOC

Agenda

- 1) Preliminaries
  - 1.1) Introduction to Turbine and Maven
  - 1.2) Maven and M.E.T.A. installation
  - 1.3) „Hello World“
- 2) Turbine Overview
- 3) Turbine example: ApacheFaces

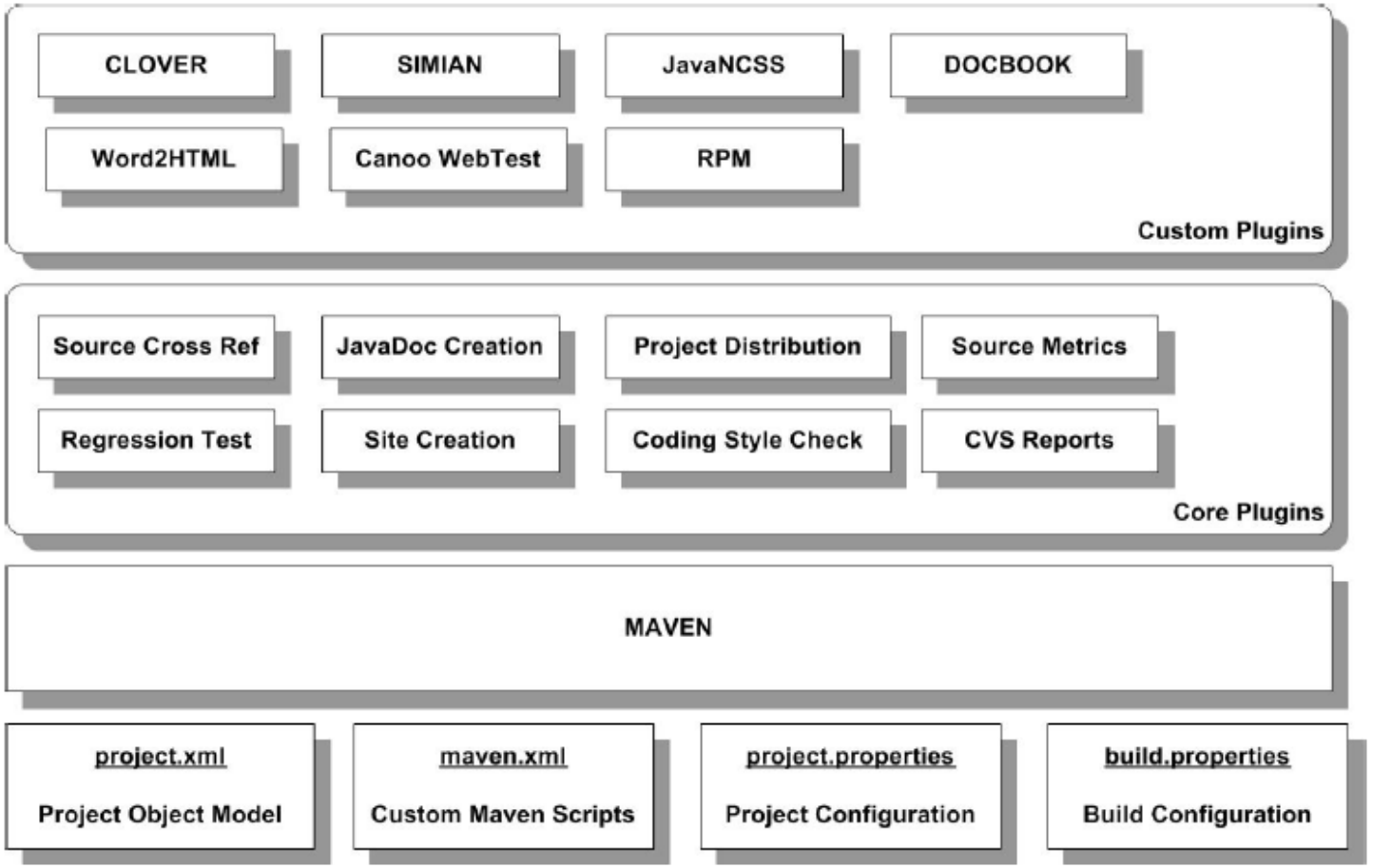
Part 1.1

Introduction to Maven and Turbine

Introduction: Apache Maven

- Integrated Project management tool
- “ant on steroids”
- Project descriptor (project.xml)
- Definition of custom build goals (maven.xml)
- Pre- and Post-Goals control build sequence
- Plugin-oriented approach
- Plugins are written in Jelly
- ~ 90 plugins included with Maven itself

Apache Maven Architecture



(Siegfried Götschl / it20one)

Apache Maven: How to get it

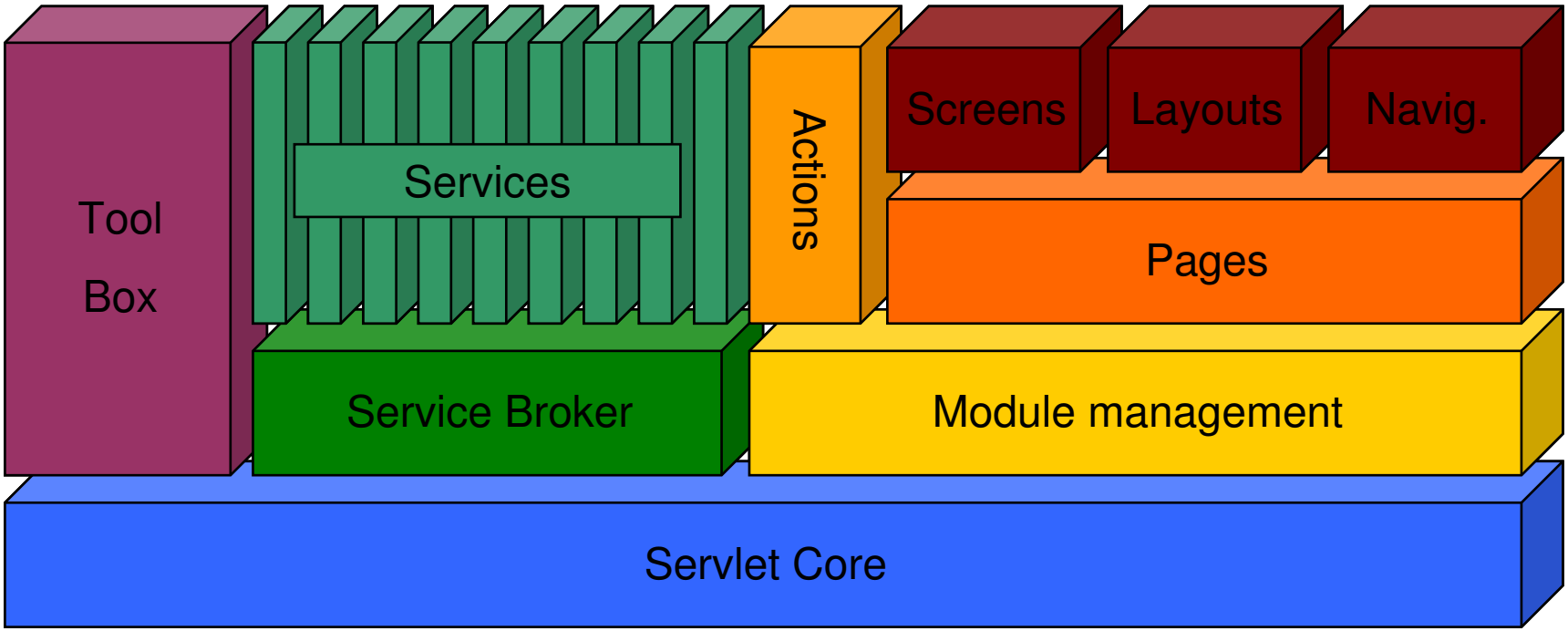
- Integrated artifact repository (jars, plugins)
- Web repo at http://www.ibiblio.org/maven/

- Current release version is 1.0.2
  (1.1 is Beta level, 2.0 is Alpha level)
- Get it from http://maven.apache.org/
- Binary and Source versions available
- Even a Windows Installer!

Introduction: Jakarta Turbine

- „A platform for building applications, not just running them“
- Toolbox for building web applications
- 100% pure Java, JDK 1.3+
- J2EE compilant, servlet based
- MVC oriented

Jakarta Turbine: Architecture



Jakarta Turbine: Architecture

- View and Controller Framework
- Service framework with ~20 services
- „Tool“ infrastructure
- Utility classes

- Jakarta Velocity to render View templates
- DB Torque for persistence

Jakarta Turbine: How to get it

- http://jakarta.apache.org/turbine/
- Current „official“ release is 2.3.1. A new major relase with architectural changes (2.4) is under development
- The Turbine project is looking for contributors helping with the code and the docs!

Turbine Development Kit

- Turbine up to 2.3 knew only the “TDK” (Turbine Development Kit)
- Large collection of ant scripts that in the end spawned the Maven project
- No official version since Turbine 2.2
- Heavy (includes servlet container and jars)
- Rigid (path structure dictated by the TDK)
- Underdocumented

M.E.T.A.

- Maven Environment for Turbine Applications
- Custom Maven plugin
- First release with Turbine 2.3.1 (supporting 2.3 and 2.3.1)
- Lightweight (downloads lots of stuff from the 'net, though)
- IDE support (tested with Eclipse)
- Docs: http://jakarta.apache.org/turbine/meta/

Part 1.2

Maven and M.E.T.A. installation

The M.E.T.A. environment

- Custom plugin for Maven
  - Setup of a new Turbine-based application
  - Compilation of Turbine and Torque based code
  - Generation of necessary SQL files for persistence
  - Deployment and WAR generation

- Can be used with “old fashioned editor” (OFE) environment or IDE

Prerequisites

- A J2SDK, supporting Java 1.3 or Java 1.4
- A servlet container (Unlike the TDK, you are free to choose one)
- A database for persistence.
- Something for code editing. Either an IDE oder an editor. Or both.

Software used in the tutorial

- Sun J2SDK 1.4.2 – http://java.sun.com/
- Apache Tomcat 5.0.x – http://jakarta.apache.org/tomcat/
- MySQL 4.1 – http://www.mysql.org/
- Eclipse 3.0 – http://www.eclipse.org/
- GNU emacs – http://www.gnu.org/software/emacs/

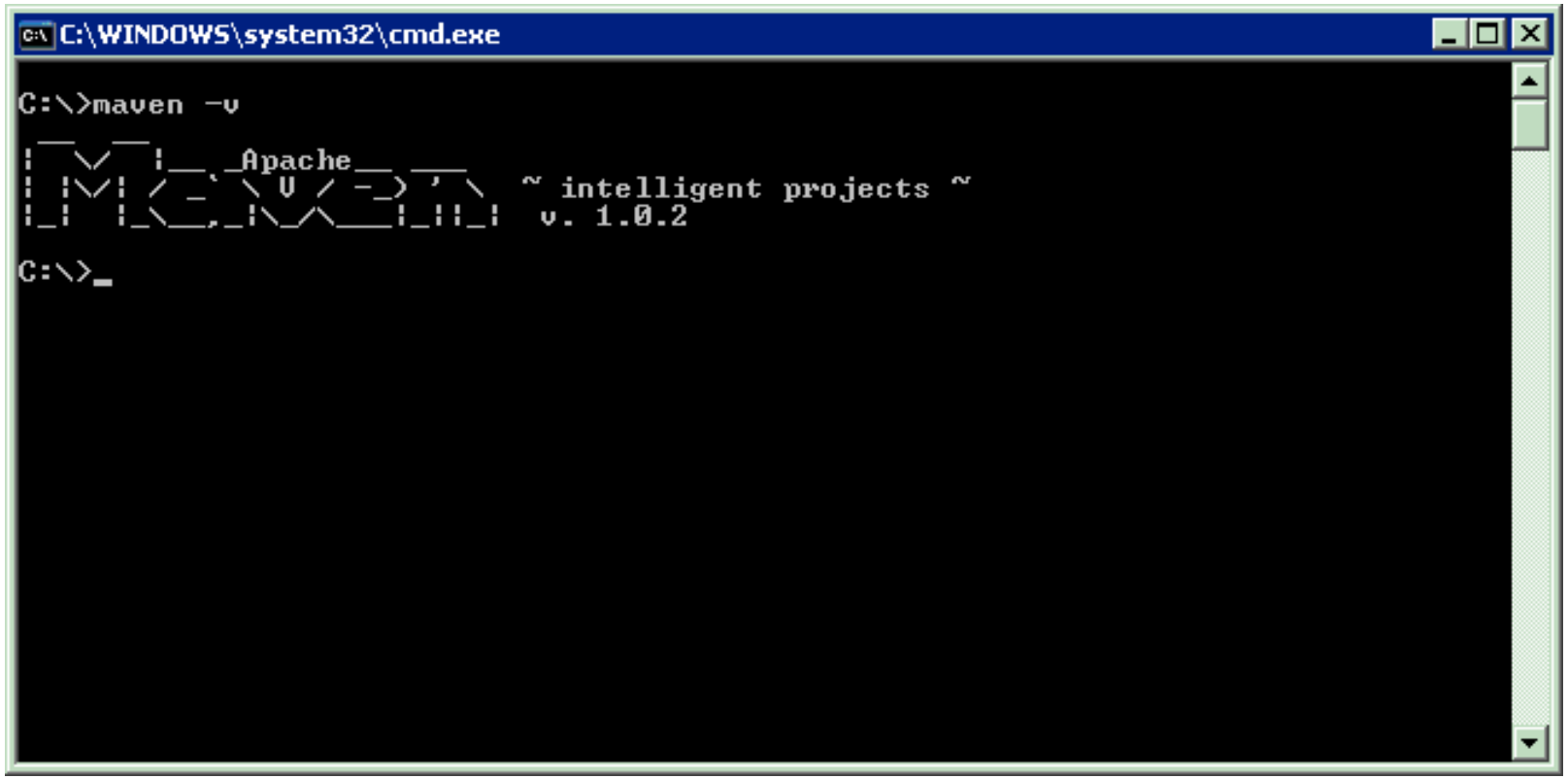
Installing Maven

- http://maven.apache.org/start/download.html
- Get **maven 1.0.x**, don't even think about the 1.1 or 2.0 alpha / beta versions.
- Windows
  - The Installer works nicely
- Unix / Linux
  - Create ~/.maven and unpack there

Caveat

- **`JAVA_HOME`**
  - On Windows: Settings -> System -> Environment
  - On RPM based Linux: Use the excellent JPackage RPMs which do a really good job to integrate Java seamlessly
- **`MAVEN_HOME`**
  - On Windows: Windows Installer does it right
- global **`build.properties`** file
  - Don‘t forget **`maven.appserver.home`** setting!
  - On Windows: Make sure to use forward („/“) slashes
- Add **`MAVEN_HOME/bin`** to your **`PATH`**

Does Maven work?

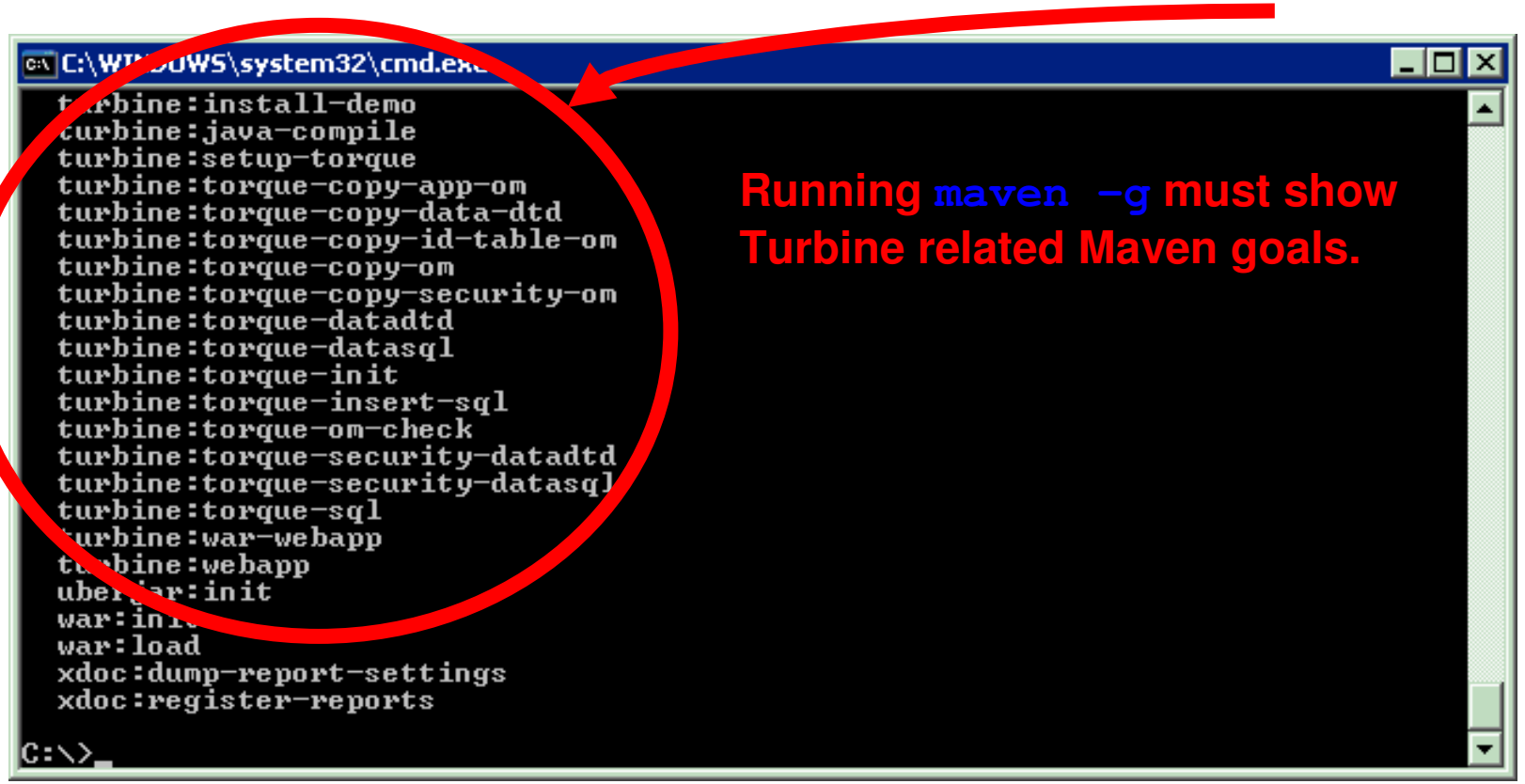


Installing M.E.T.A.

- Maven can load M.E.T.A. directly from the central repository at ibiblio

```
maven -DartifactId=maven-turbine-plugin \
    -DgroupId=turbine \
    -Dversion=1.2 plugin:download
```

Did it install correctly?



Ready to go!

- The M.E.T.A. workbench consists of
  - Java SDK
  - Servlet Container
  - Maven
  - M.E.T.A. plugin
- Optional components
  - IDE
  - Database

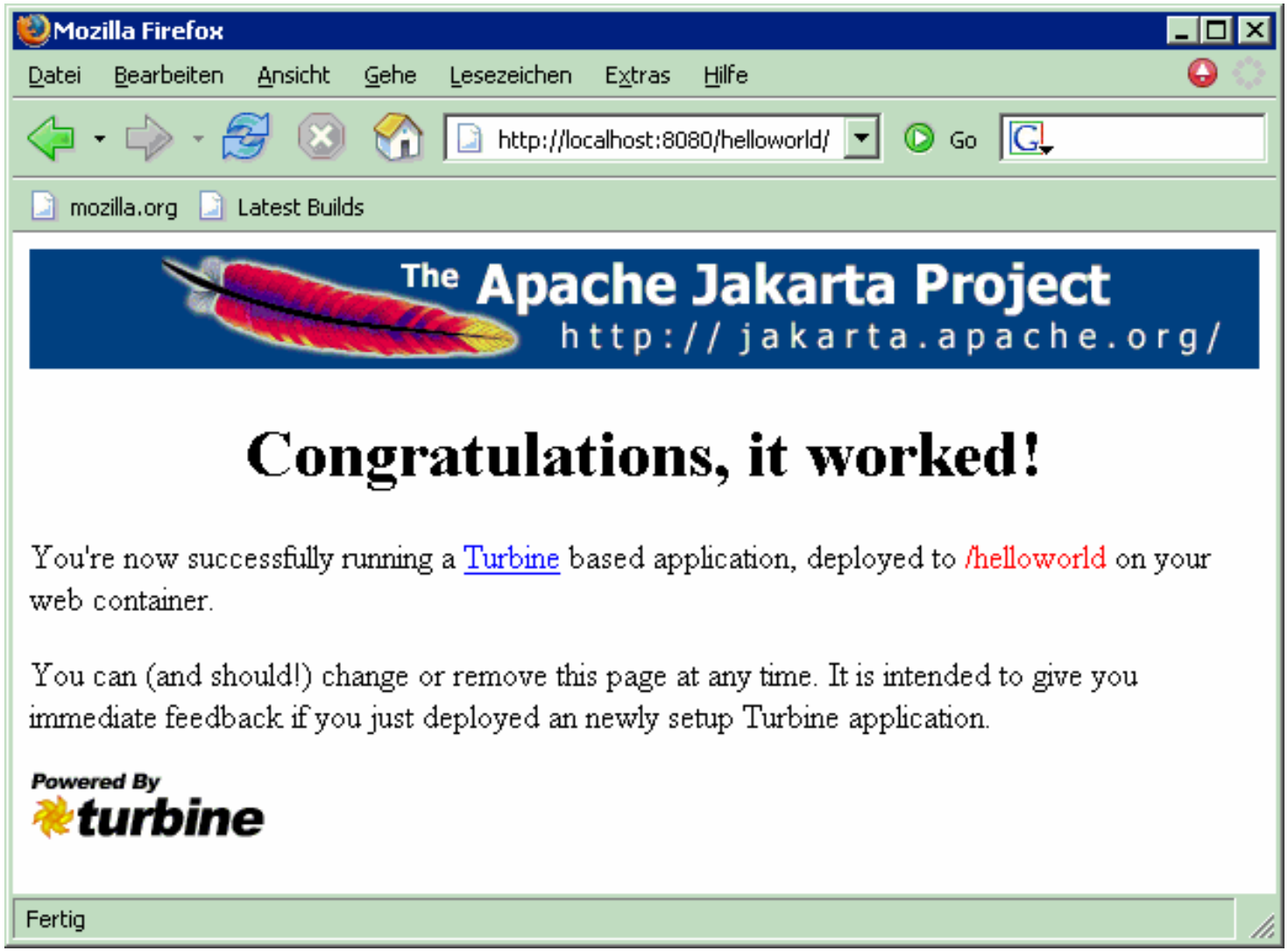
Part 1.3

„Hello World“

Instant “Hello World”

- Setup build environment
  - `maven -Dturbine.app.name=demo turbine:setup`
- Compile application
  - `maven java:compile`
- Deploy it to the web container
  - `maven turbine:deploy`
- Run the container

“Hello, World”


Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
http://localhost:8080/helloworld/
Go
mozilla.org
Latest Builds
The Apache Jakarta Project
http://jakarta.apache.org/
Congratulations, it worked!
You're now successfully running a Turbine based application, deployed to /helloworld on your web container.
You can (and should!) change or remove this page at any time. It is intended to give you immediate feedback if you just deployed an newly setup Turbine application.
Powered By
turbine
Fertig

What has happened?

- `turbine:setup` generates a new application
- `java:compile` builds the .class files
- `turbine:deploy` installs the web application
- Skeleton contains a few demo pages
- “normal” development mode using an editor (“for the emacs folks”)

- Note: no `setup.properties` file necessary

Caveat

- Make sure your Maven setup is correct!
- You **must** have Internet connectivity when building your project for the first time
- The “missing libraries” problem
- Starting / Stopping your Servlet Container

M.E.T.A. goals in “normal” mode

- Skeleton setup
  - `turbine:setup`
- Development cycle
  - `java:compile`
  - `turbine:deploy`
- Bind WAR for deployment
  - `turbine:war`
- Documentation
  - `site`

Using an IDE – “inplace” mode

- “inplace” development mode
- Custom **`setup.properties`** file necessary:

```
turbine.app.name              = inplace
turbine.app.subdir            = true
turbine.app.flavor            = turbine-2.3.1
turbine.app.om.layer          = torque
turbine.plugin.mode           = inplace
turbine.plugin.inplace.dir = tomcat
```

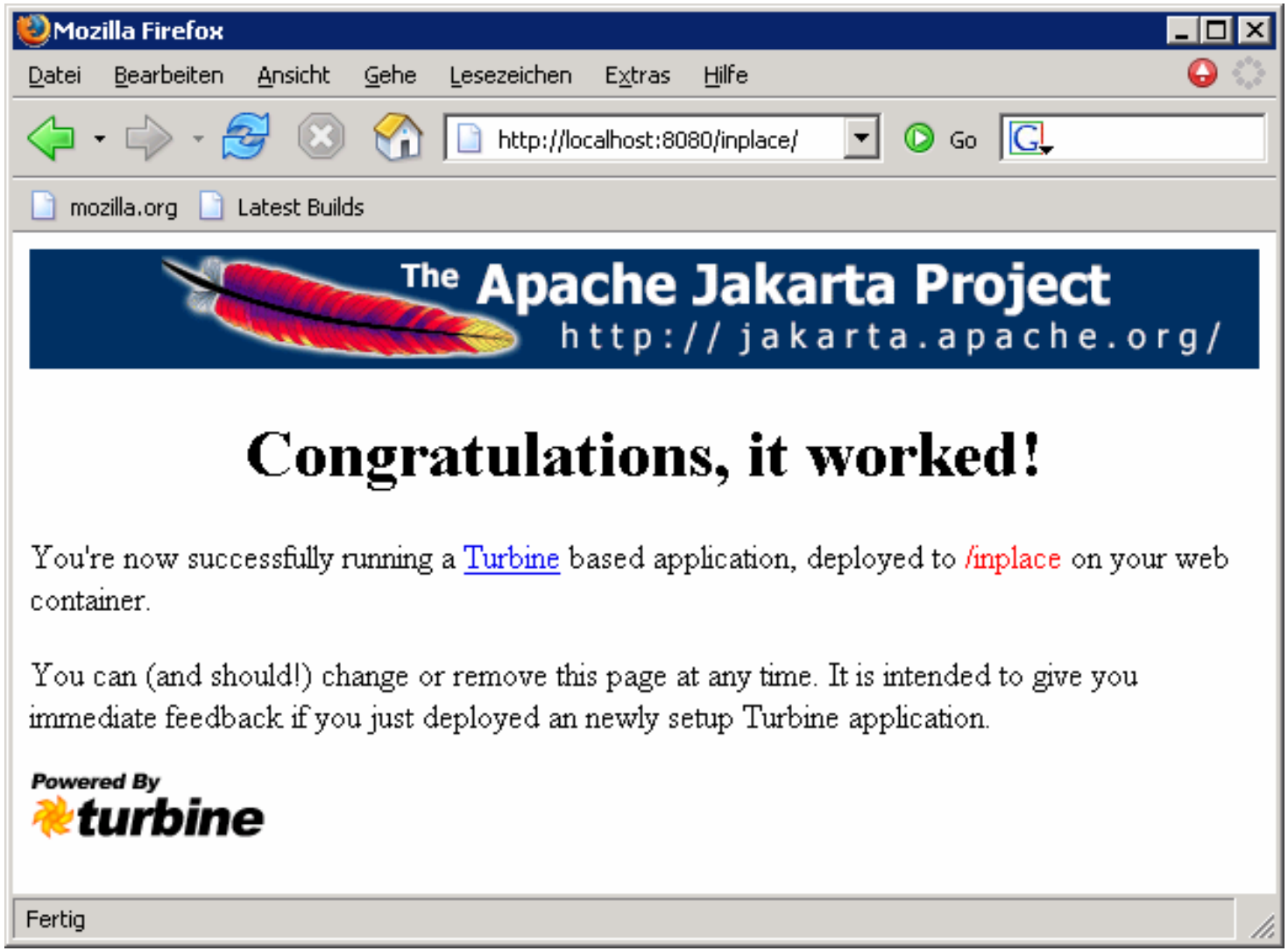
Setup for Eclipse

- Setup build environment
  - `maven turbine:setup`
- Generate persistence classes
  - `maven torque:om`
- Fetch all necessary libraries
  - `maven turbine:install-libs`
- Generate Eclipse Configuration
  - `maven eclipse`
- Import project into Eclipse
- (optional) Add Information for Tomcat plugin

Eclipse Tomcat plugin

- Eclipse plugin for Tomcat Start/Stop
- Also integrates configuration with Project
- Download from
  http://www.sysdeo.com/eclipse/tomcatPlugin.html

“Hello, World”, inplace style



What has happened?

- `turbine:setup` generates a new application
- `torque:om` builds persistence layer sources
- `turbine:install-libs` fetches the libs
- `eclipse` generates .project and .classpath
- Skeleton contains a few demo pages
- “inplace” development mode for IDE users
- Not the default, a `setup.properties` file must be present!

M.E.T.A. goals in “inplace” mode

- Skeleton setup
  - `turbine:setup`
- Eclipse setup
  - `turbine:install-libs`
- Update source code
  - `torque:om`
- Documentation
  - `site`

“normal” vs. “inplace” Mode

- Maven builds project
- Development cycle
  - `java:compile`
  - `turbine:deploy`
- Webapp location outside source tree
- Autogenerated classes are automatically updated

- IDE builds project
- Maven prepares project
  - `torque:om`
  - `turbine:install-libs`
- Webapp location inside source tree
- Autogenerated classes must be updated
  - `torque:om`

What about SQL code?

- M.E.T.A. also builds SQL code if a persistence layer is configured
  - `turbine:sql` – builds SQL code
- Torque specific support:
  - `torque:insert-sql` – insert SQL into database
  - `torque:create-db` – create the database

End of Part 1

- What have we done?
  - set up our workbench
  - installed M.E.T.A.
  - talked about development modes
  - old-fashioned and IDE-based “Hello World”
- What have we not done?
  - Talked about APIs or programming
  - Wrote a single line of code. No Java, XML, …

five minute break

Part 2

Turbine Overview

Why Turbine?

- Well suited for “spontanous development” (aka “agile processes” aka “hacking”)
- Turbine applications can “grow”
- The Turbine concepts are easy to understand
- A Number of helper classes for “boring” tasks

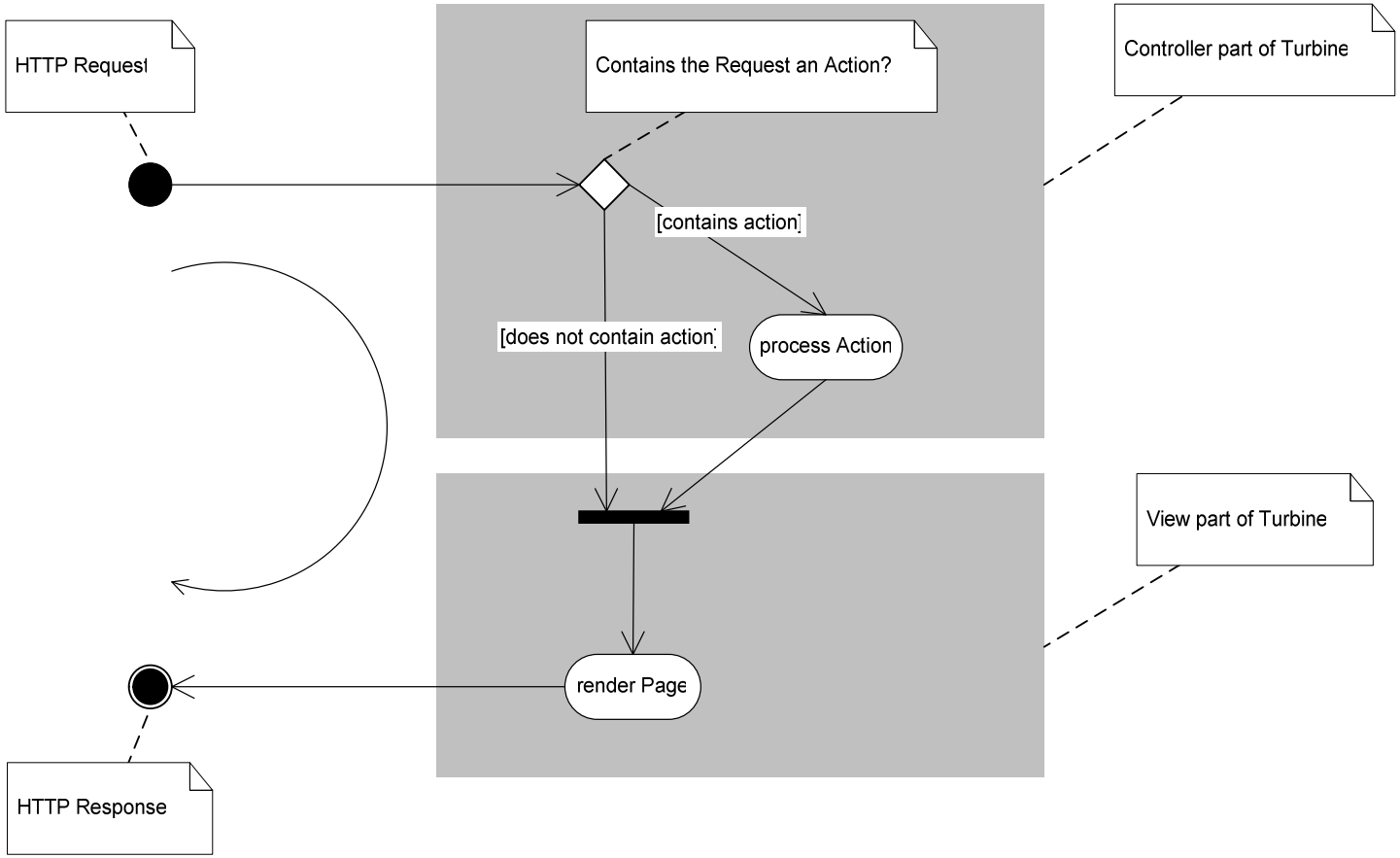
Turbine buzzwords

- **Modules**. Generic term for View parts and Actions. Modules are organized in Java packages.
- **Tool**. Part of the tool box provided by Turbine
- **Action**. Part of the Controller, responsible for changing model state.
- **Service**. Plug-in part of the framework

Model 2+1

- blurs the MVC concept by integrating View and Controller
- “paradigm” for writing Turbine applications
- expression coined by Jon S. Stevens
- based on MVC (Model 2)

Turbine MVC

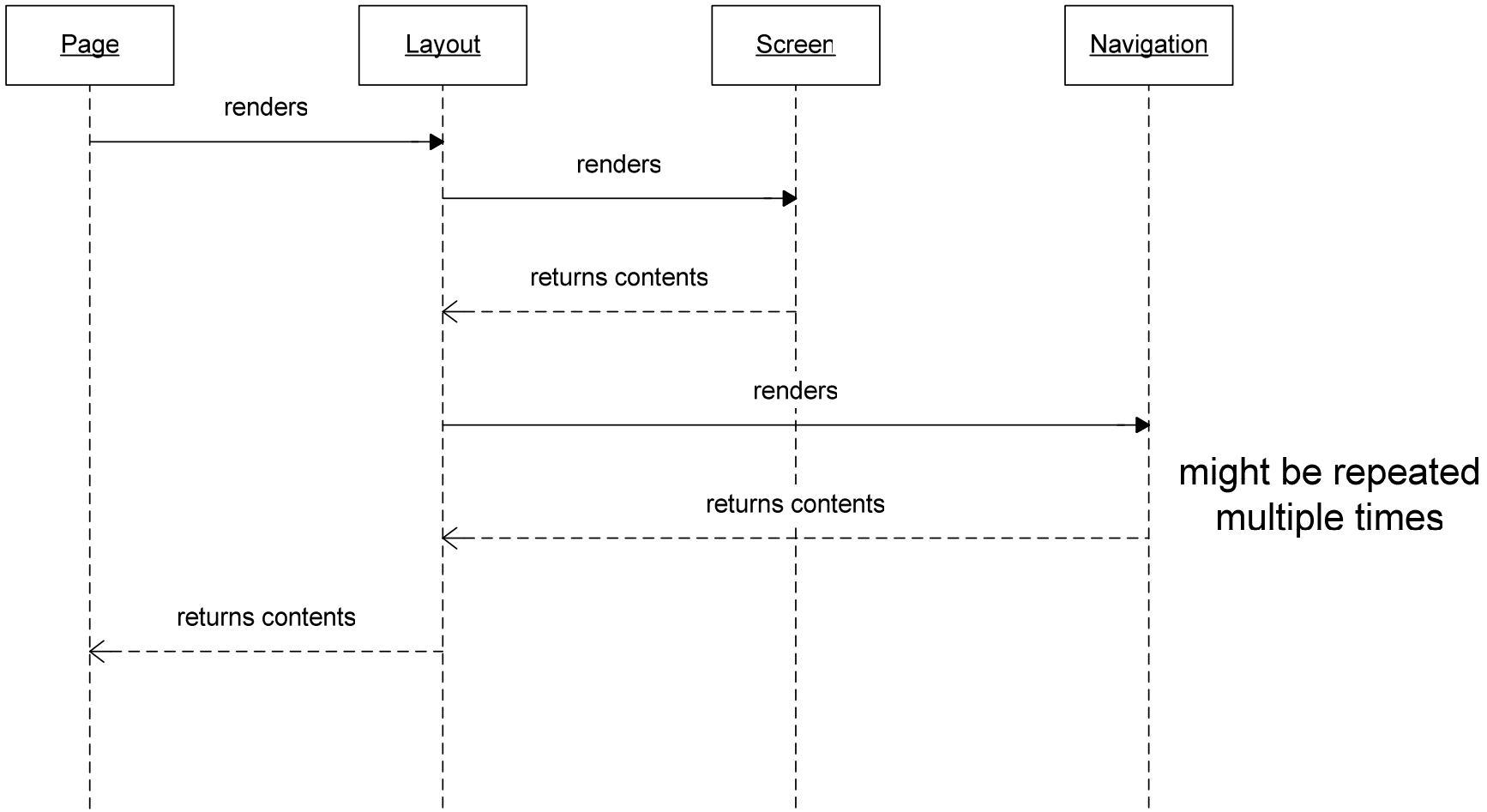


You are not expected to be able to read this slide from the last row in the audience. This is why you got hand-outs.
If you can, call NASA at +1 202 358 0001 to become a test pilot instead of a Java Hacker

Turbine View Model

- The Turbine View is composed from
  - Page “Everything you see”
  - Screen “Content of the page”
  - Navigation “Top, Bottom, Menu”
  - Layout “Where to put everything”
- Each part of the View is a Module

Turbine View composition



Turbine View quirks

- Most web frameworks use a Templating solution or JSPs for the View
- **Turbine uses Java classes for Page, Layout, Screen and Navigation**
- Turbine provides classes that implement templating with Velocity or JSPs
- Using Java classes as View makes things like PDF rendering screens easy

Turbine View Classes

- View classes are controlled by the module manager (Assembler Broker)
- Java class names are composed from the module names and packages
- Default module package is **`org.apache.turbine.modules`**
- additional packages can be added in the configuration

Templating - Velocity

- A View should be built from web pages, JSPs or Templates, not Java classes
- Turbine leans heavily towards Templates and uses the Velocity templating engine
- Building the View with templates needs “glue” for calling the templating engine

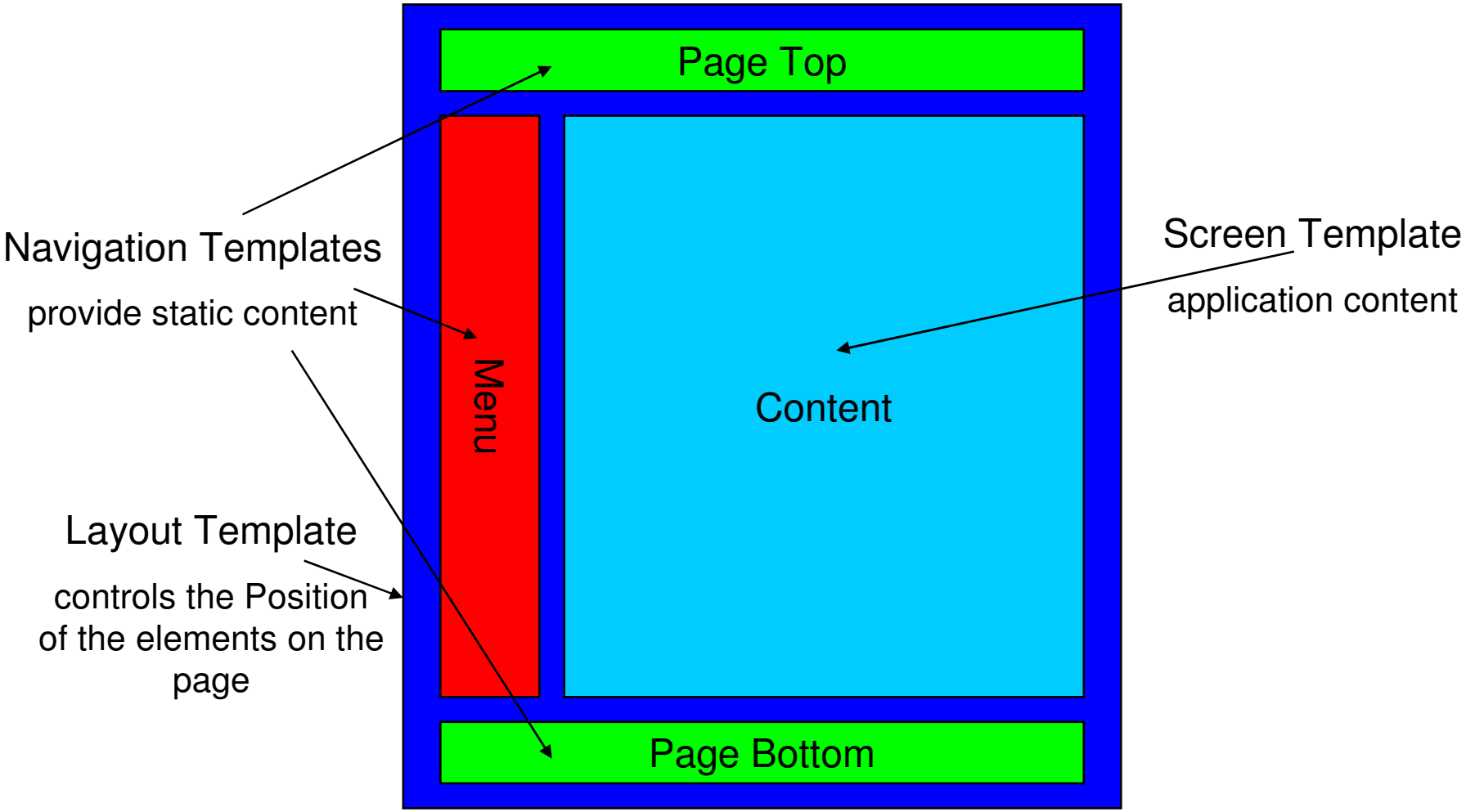
Velocity in a nutshell

- “The other ugly duckling”
- Designed to replace WebMacro
- Simple notation for conditionals and loops
- Contains macro definition and execution
- Can load its templates from many sources
- Provides Context between Java and Template code
- Integrated bean access in template code

Velocity and Turbine

- Turbine provides classes for Velocity integration:
  - Pages: VelocityPage
  - Screens: VelocityScreen, VelocitySecureScreen
  - Layouts: VelocityOnlyLayout, VelocityDirectLayout
  - Navigation: VelocityNavigation
- These classes are used as defaults

Velocity View composition



Example: A Layout Template

```
<html>
<head/>
<body>
  <table>
    <tr><td colspan="2">$navigation.setTemplate("Top.vm")</td></tr>
    <tr>
      <td>$navigation.setTemplate("Menu.vm")</td>
      <td>$screen_placeholder</td>
    </tr>
    <tr><td colspan="2">$navigation.setTemplate("Bottom.vm")</td></tr>
  </table>
</body>
</html>
```

Navigation Template References

Screen Template Reference

Requesting a Page

Pages are requested by providing the “template” CGI parameter

```
http://localhost:8080/helloworld/app/template/Demo.vm
```

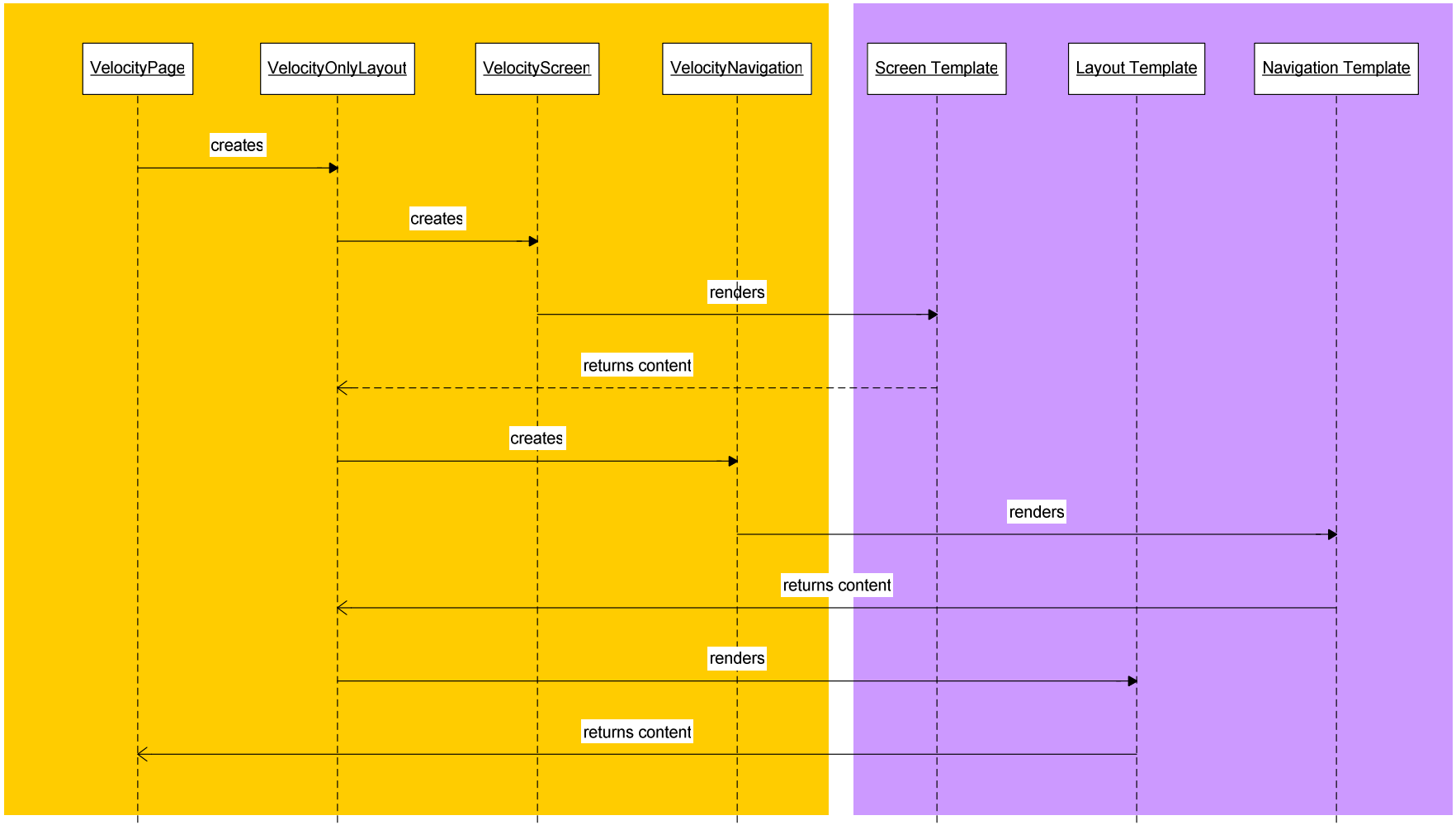
Templates name parts are separated with a comma

```
Demo.vm
demo,Demo.vm
complex,page,Example.vm
```

Template names are not file paths!

- Prior to T2.3, template naming was a mess
- T2.3 introduced the two golden rules for templates:
  - “Template Names never contain slashes”
  - “Template names are no paths. They’re not absolute and have no leading slash”
- A template name can resolve to a very different filename

View rendering with Velocity



Matching templates

- Page class, Screen and Layout templates are found by a search mechanism
- Java classes and Templates are looked up using various search patterns
- Each search pattern has a default value
- Most search mechanisms work in a hierarchical fashion

Matchmaking

- When requesting a Page, Turbine fetches Layout, Screen and Navigation

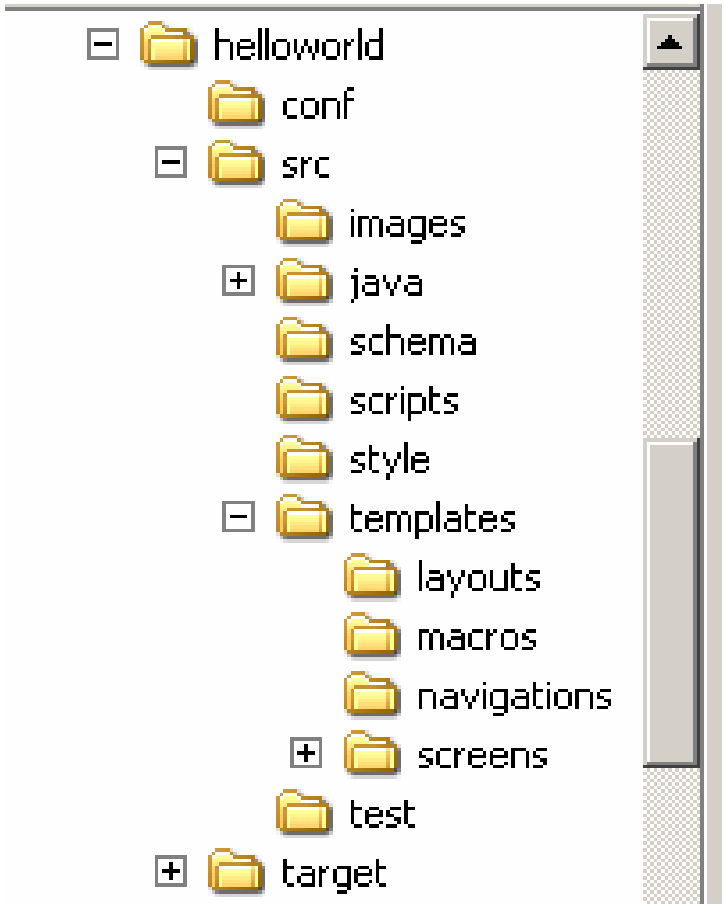
- Finding a Layout template:

Request:	**`demo,page,Page.vm`**
1. match:	**`demo,page,Page.vm`**
2. match:	**`demo,page,Default.vm`**
3. match:	**`demo,Default.vm`**
4. match:	**`Default.vm`**

Template Files

- File lookup through Velocity
- Separate trees for all template types
- Templates are kept in sub directories

- This is just the default!



Dynamic content

- Until now we have seen only static content
- A Web application (of course) needs active content
- And we haven’t programmed a single line of Java code yet!
- But there were some strange place holders starting with a dollar sign…

The Velocity context

- Vehicle between templates and Java code
- The context can contain arbitrary Java objects referenced by a key (hash table)
- Velocity provides access to Bean getters with a short hand notation
- Velocity can call any method on an object in the context.

Getting Stuff in the Context

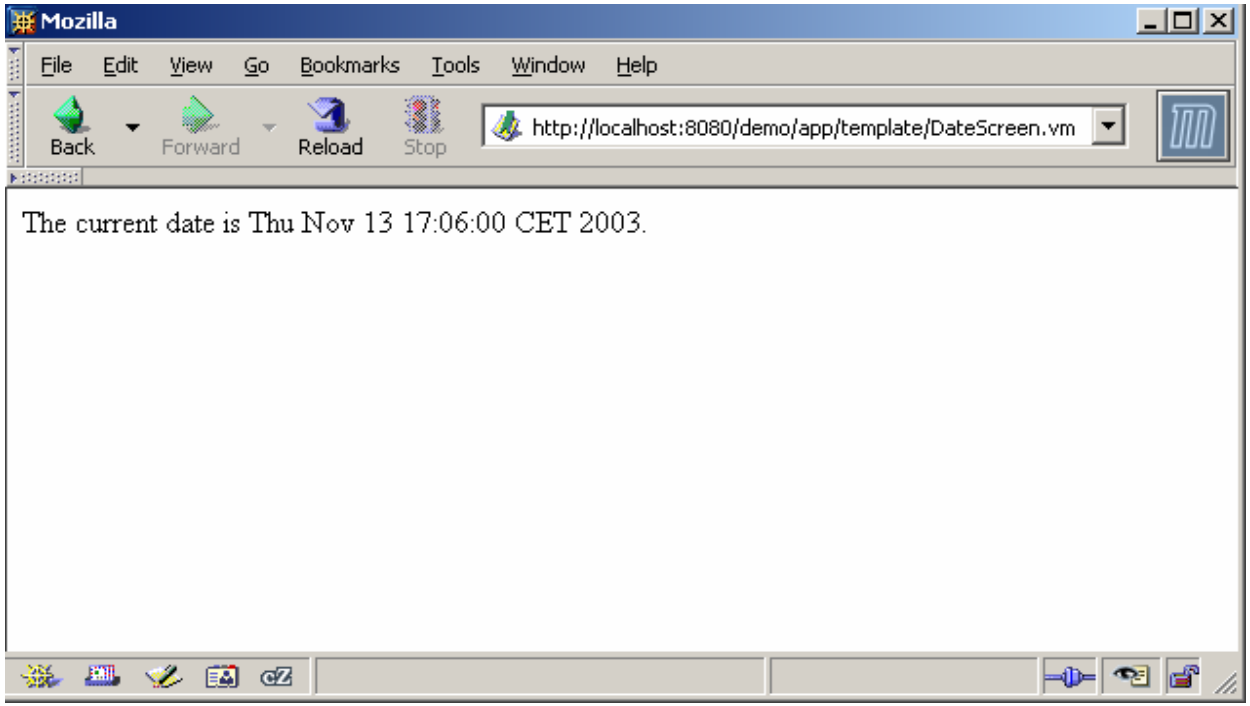
- All Screens in the View are rendered by Java classes
- A mechanism similar to the template lookup exists for the Java classes
- And, of course, we can provide our own class…

Example: A Screen class

```
public class DateScreen
   extends VelocityScreen
{
   public void doBuildTemplate(RunData data,
                               Context context)
       throws Exception
   {
       Date d = new Date();
       context.put("date", d);
   }
}
```

Example: Screen Template

```
The current date is $date.
```



Side Track: RunData Object

- Turbine instantiates a RunData object for each request/response cycle
- This object gives access to user, session and Turbine related information
- It is only valid for one request cycle

“Don’t keep a reference in application objects unless you know exactly what you’re doing!”

Screen classes make sense?

- Writing a class for each Screen is much work
- Lots of Code duplication!
- Code and Template must be kept in sync
- All objects must be put in the context by every screen

Screen classes are the push model

Turbine Tools

- Tools are small Java classes that implement a turbine-specific tool interface
- Tools are managed and added to the context by Turbine
- Tools have defined scopes and lifetime

Tools are the pull model

Tool Scopes

- global — global for all sessions
- request — per request cycle
- session — per user session
- authorized — per users session, after login
- persistent — per user, is put in storage

Example: Date Tool

```
public class DateTool
    implements ApplicationTool
{
    private Date d = null;

    public DateTool() { /* empty */ }

    public void init(Object data)
    {
        d = new Date();
    }

    public void refresh() { /* empty */ }

    public String getDate() {
        return d.toString();
    }
}
```

Application Tool Interface

Custom Tool Code

Activating the Date Tool

- Adding the tool to Turbine configuration

```
# Request Tool. Refreshed with every request cycle
tool.request.requestDateTool = de.intermeta.demo.tools.DateTool

# Global Tool. Instantiated only once
Tool.global.globalDateTool = de.intermeta.demo.tools.DateTool
```

- Usage

```
<p>Request Tool: The current date is $requestDateTool.getDate()</p>

<p>Global Tool: The current date is $globalDateTool.getDate()</p>
```

Predefined tools

- Turbine contains a number of pull tools
- Some tools are even activated by default
  - **`$link`** builds URI for linking template pages
  - **`$content`** URI for container provided pages like images or style sheets
  - **`$page`** controls HTML HEAD and BODY attributes.

- The RunData object is available as **`$data`**

Using predefined tools

```
$page.setBgColor("yellow")

<a href="$link.setPage("demo,Target.vm")">Go to the Target</a>

<img src="$content.getURI("images/apache.png")">
```

Resulting HTML code

```
<html>
  <head><title></title></head>
  <body BGCOLOR="yellow">
    <a href="/demo/app/template/demo%2CTarget.vm">Go to the Target</a>
    <img src="/demo/images/apache.png">
  </body>
</html>
```

Tool lifecycle

- Tool interface methods
  - `init()`
  - `refresh()`

- `init()` is called at tool initialization

- `refresh()` is called each time the tool is put to the context

Turbine Actions

- In Model 2+1, the Controller is part of the Turbine servlet
- Turbine provides Actions to map Controller actions to Java classes
- Actions are modules which are managed by the Assembler Broker
- There are no default actions

Requesting a Turbine Action

- Actions are requested by providing the action parameter

```
http://localhost:8080/helloworld/app/action/LoginUser
```

- The **`$link`** Tool contains a few convenience methods for building Action requests
- Every action has its own class

Example: Action

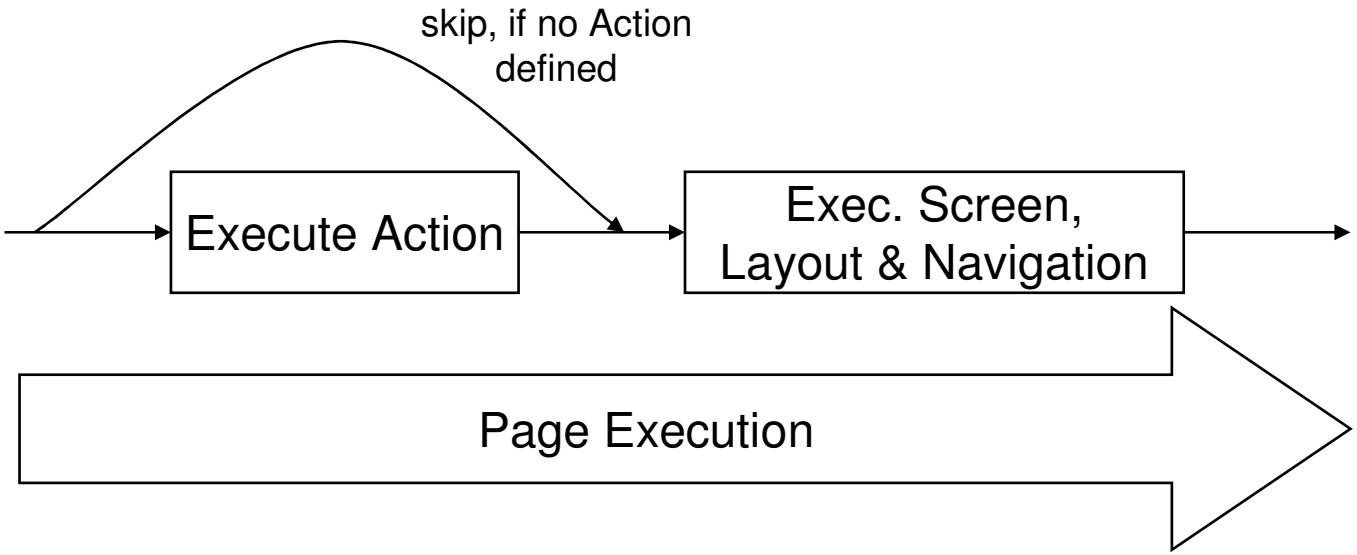
```
import org.apache.turbine.modules.actions.VelocityAction;
import org.apache.turbine.util.RunData;

import org.apache.velocity.context.Context;

public class DemoAction
    extends VelocityAction
{
  public void doPerform(RunData data, Context context)
     throws Exception
  {
    String msg = "Action was executed!";
    context.put("done", msg);
  }
}
```

Executing Actions

- Due to historic reasons, Actions are actually part of the Page
- You can't change the page from an Action!



Action Events

- Grouping actions together in classes is done with ActionEvents
- Action Events are very useful for Forms
- Action Event avoid having lots of classes with just one method
- The **`$link`** tool provides methods for dealing with Action Events

Form Submission

- Action Events are called by providing special named parameters to a request
- The method name is prefixed with “eventSubmit_”

```
<form method="post"
 action="$link.setPage("FormResult.vm").setAction("FormAction")">
 Enter a value: <input type="text" size="10" name="value"><br/>
 <input type="submit" value="Cancel" name="eventSubmit_doCancel">
 <input type="submit" value="Enter" name="eventSubmit_doEnter">
</form>
```

Example: Action Event

```
public class FormAction
        extends VelocityAction
{
[…]
    public void doCancel(RunData data, Context context)
        throws Exception
    {
        data.setScreenTemplate("Cancel.vm");
    }

    public void doEnter(RunData data, Context context)
        throws Exception
    {
        context.put("value", data.getParameters().getString("value", ""));
    }
}
```

Action Event in a link

- **`$link`** provides support for Action events

```
<a href="$link.setActionEvent("FormAction", "doEnter")">
```

- Resulting HTML Code

```
<a href="/demo/app/action/FormAction/eventSubmit_doEnter/1">
```

Turbine Services

- Singleton based architecture
- Most parts of the Turbine core are services
- Only one Service object per Service for all sessions and users
- Service broker provides management for startup and shutdown

Existing Services

- Services used by the HelloWorld application
  - AssemblerBroker (Module Management)
  - RunData Service (Run Data Management)
  - Template Service (Template Lookup)
  - Velocity Service (Velocity Rendering)
  - Pull Service (Tools)
  - Factory & Pool Service (Object Management)

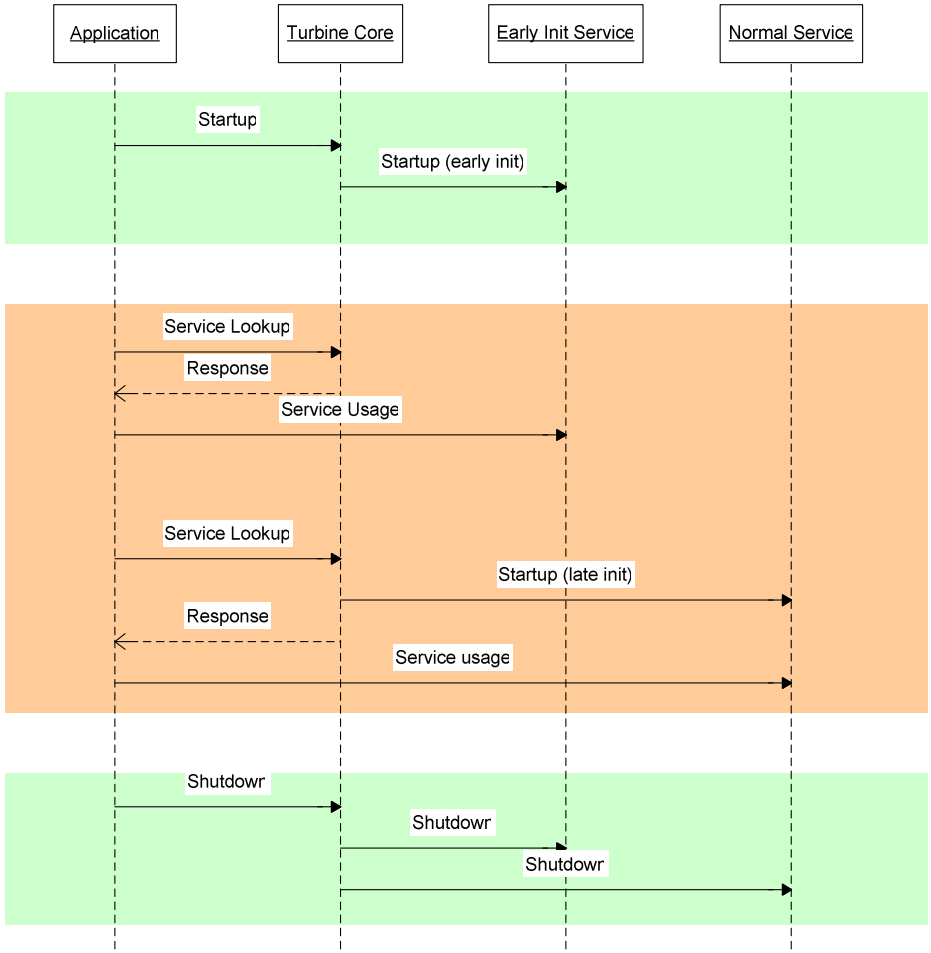
Service Management

- Services are configured in the Turbine configuration file

- “late init”: Initialization at first lookup
- “early init”: Initialization at Turbine startup

- Default is “late init”

Service lifecycle



Turbine Services

- Turbine provides ~ 20 different services
  - Security Service (User authentication)
  - Intake (Input validation)
  - Localization
  - Upload (file uploading)

Accessing Turbine Services

- Most Turbine services provide a static Façade for calling its methods
- Using the Façade hides the Service lookup and makes the calling code more readable
- Each façade should provide a **`getService()`** method for fetching a reference to the Service object

Example: Accessing a Service

```
public class CryptoTool
  implements ApplicationTool
{
  private CryptoService cs = null;
[…]
  public void init(Object data)
  {
    cs = TurbineCrypto.getService();
  }
[…]
  public String encrypt(String value)
    throws Exception
  {
    CryptoAlgorithm ca = cs.getCryptoAlgorithm("MD5");
    return ca.encrypt(value);
  }
}
```

Example: Service skeleton

```
import org.apache.turbine.services.BaseService;

public DemoService
  extends BaseService
{
  public void init() throws InitializationException
  {
    setInit(true);
  }

  public void shutdown()
  {
    setInit(false);
  }

  [… add your service methods here …]
}
```

Activating a Service

- Services are added in the Turbine Configuration

```
services.DemoService.classname=de.intermeta.demo.DemoService
```

- For early initialization, an optional parameter can be added

```
services.DemoService.earlyInit = true
```

Turbine parameter passing

- Turbine allows an application to use three types of parameters
  - CGI GET (…?param=value)
  - CGI POST (stdin on a POST request)
  - PATH_INFO (…/param/value)
- All three methods are equal
- PATH_INFO allows bookmarking
- PATH_INFO params must be in pairs!

Getting Parameters

- All CGI and PATH_INFO parameters are pulled together
- Multiple params are available as Arrays
- The RunData object provides access to the parameters
- The values can be queried as various types

Getting Parameters

```
String s = data.getParameters().getString("val");
String s = data.getParameters().getString("val",
  "default");

int i = data.getParameters().getInt("val");
int i = data.getParameters().getInt("val", 0);
int [ ]  i = data.getParameters().getInts("val");

boolean b =
  data.getParameters().getBoolean("bool", false);

Set allParamNames = data.getParameters().keySet();
```

Turbine Configuration

- Turbine uses commons-configuration to read its configuration
- Default is properties based configuration from **`TurbineResources.properties`** file
- multiple lines or comma-separated values are provided as multiple values for a property
- M.E.T.A. separates application and core configuration

End of Part 2

- What have we done?
  - Learned about the main parts of Turbine
  - View with Page, Screen, Navigation, Layout
  - Tools
  - Pull and Push model
  - Controller with Action and Action Events
  - Services

five minute break

Part 3

Turbine example: ApacheFaces

About ApacheFaces

- An application to be able to “match faces to names”
- Written as an entry to the ApacheCon 2004 Derby programming contest
- Written in about six hours time during AC2004
- Made 2nd place.

Any questions?

Where to go from here

- The Turbine home page
  - http://jakarta.apache.org/turbine/
- All materials from this talk are available at
  - http://henning.schmiedehausen.org/turbine/

Thanks a lot for your attention