CICS and Web 2.0 – End to End Development

Dennis Weiand
IBM

Wednesday, August 4, 2010
Session #6924

SHARE in Boston

Abstract

- As a whole new generation of web-based social networks, collaborative, and interactive technology tools have emerged, businesses are beginning to deploy them within the enterprise. These tools enable improved decision making and faster response to emerging situations by providing rapid and intuitive access to content and social connections. This speaker will describe their experience integrating Web 2.0 technologies with CICS applications, making use of SupportPacs and technology previews available for use with CICS TS V3.

Trademarks

- The following terms are trademarks of the International Business Machines Corporation or/and Lotus Development Corporation in the United States, other countries, or both:
  - Redbooks(logo)™, AIX®, alphaWorks®, CICS®, DB2®, IBM®, IMS™, Informix®, MQSeries®, VisualAge®, WebSphere®
- The following terms are trademarks of other companies:
  - Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation.
  - Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc.
  - CORBA, CORBAServices, and IIOP are trademarks of the Object Management Group, Inc.
  - UNIX is a registered trademark of The Open Group in the United States and other countries.
  - Other company, product, and service names may be trademarks or service marks of others.

Notices

- This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this presentation in other countries.
- INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PRESENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILTY OR FITNESS FOR A PARTICULAR PURPOSE.
- This information could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this presentation at any time without notice.
- Any references in this presentation to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Agenda

- CICS Dynamic Scripting feature Pack
- Why
- What
- How
- When (now)
- What about CICS TS V3.x?
  - SupportPac CA1S for V3.2
  - No scripting in V3.1 (other than REXX)

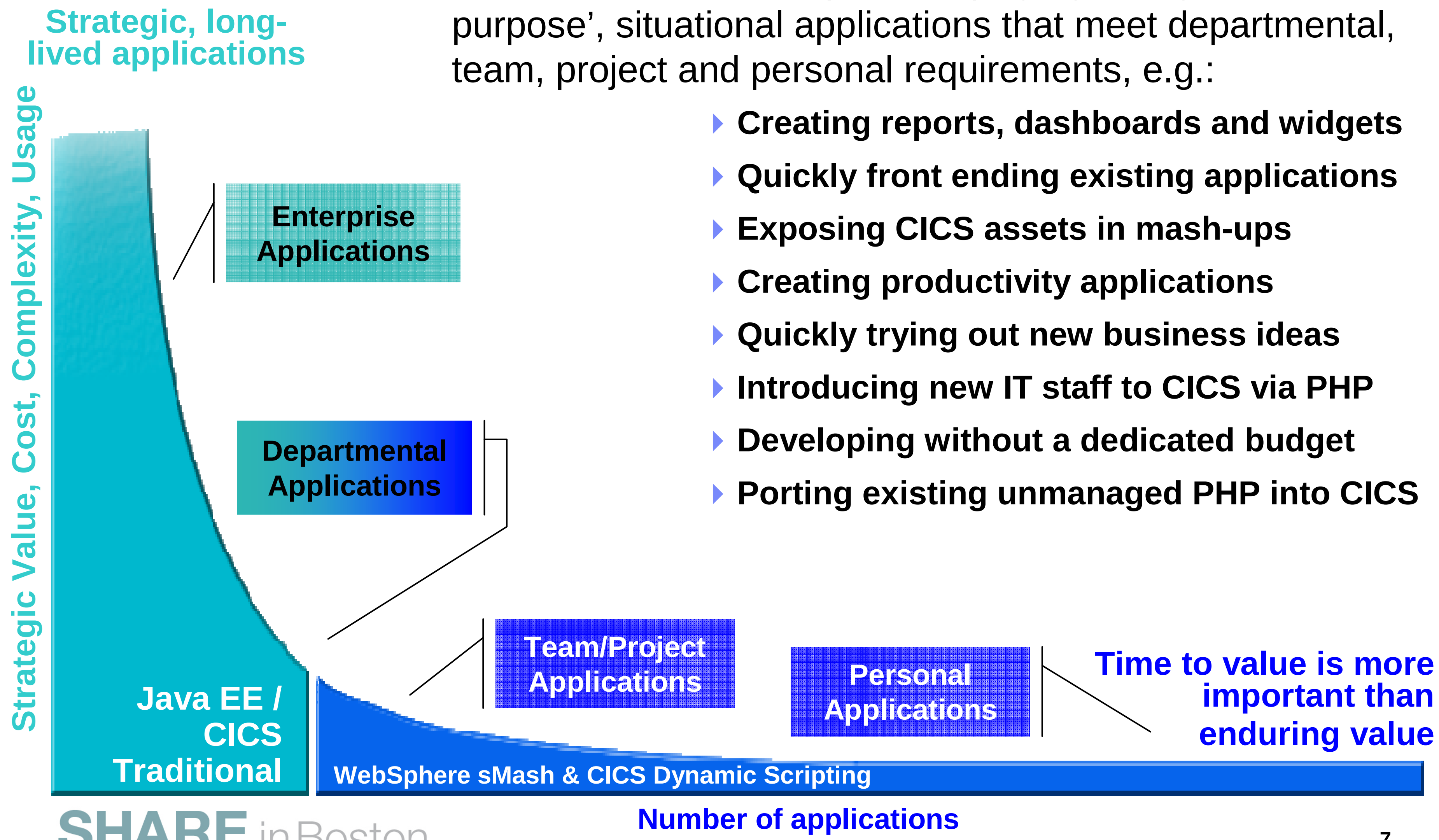
Notes:

- This presentation is primarily intended to provide a basic introduction to the CICS Dynamic Scripting Feature Pack.
- This Feature Pack became avaialble on July 22, 2010 and is a no-charge feature of CICS TS V4.1
- Since the CICS Dynamic Scripting Feature Pack is only available to CICS TS V4.1 customers, this presentation will add a few words (not many) on SupportPac CA1S which offers PHP support to CICS TS V3.2 customers
- REXX is a scripting language that has been supported for quite some time in CICS and is supported in CICS TS V4.1, V3.2, and V3.1.
- For CICS TS V3.1, only REXX is available, but no PHP or Groovy is available

CICS Dynamic Scripting

Can be used to develop and deploy lightweight, ‘fit for purpose’, situational applications that meet departmental, team, project and personal requirements, e.g.:

- Creating reports, dashboards and widgets
- Quickly front ending existing applications
- Exposing CICS assets in mash-ups
- Creating productivity applications
- Quickly trying out new business ideas
- Introducing new IT staff to CICS via PHP
- Developing without a dedicated budget
- Porting existing unmanaged PHP into CICS



Notes:

- In our typical mainframe development, we normally only address mission-critical applications. Because these applications are normally high-volume and are the life-blood of our business, we have surrounded them with procedures with tight controls that insure quality, consistency, availability and all of the other attributes needed for our main applications. These applications are normally written in CICS or WebSphere Application Server which provide industrial-strength environments for our applications.

- In addition to the main applications that handle the volume of our user interactions, there are other applications our business needs for special situations such as a sales promotion. Because of the procedures and development techniques we use, we are often not able to address application requests for the special situations (which are commonly referred to as 'situation applications').

- The demand for situational applications (sometimes referred to as the 'long tail of demand') at some companies outweighs the requests for traditional requests, but due to the procedures and development techniques we use, we don't have the time and resources to address them.

- Even if we had the 'resources', our development techniques often don't allow use to respond quickly enough to accommodate the situational application requests.

- CICS Dynamic Scripting is intended to address some of these shortcomings. CICS Dynamic Scripting, built on Project Zero technology provides a productive environment that can be used to address situational applications. CICS Dynamic Scripting is also a great way to introduce new IT staff to CICS via the Project Zero technology, and the PHP and Groovy dynamic scripting languages.

CICS Dynamic Scripting Feature Pack

- Provides **PHP and Groovy** support in CICS – agile, productive environment
- Technology from **Project Zero**, WebSphere sMash v1.1.1.3 (projectzero.org)
- Robust environment for **situational** reports, dashboards, and Web feeds
- Manageability, Scalability, and Security
- Zero Resource Model (**ZRM**) with data managed by DB2 for z/OS
- Uses CICS TS V4.1 **JVMServer** Technology
- **Situational applications** - Quickly try business ideas
- Introduce **new staff** to CICS via PHP
- Run unmanaged PHP and WebSphere sMash applications in CICS
- Easily expose CICS assets with **REST**ful interfaces
- Optional **no charge** product extension to CICS TS V4.1, June 22, 2010

Notes:

- CICS Dynamic Scripting is a Feature Pack for CICS TS 4.1.
- It embeds Zero's agile programming model into CICS on z/OS. This enables Groovy and PHP scripts to run inside CICS to handle HTTP requests. You can exploit many of the features provided by Project Zero technology to quickly and easily build custom services and applications around your CICS programs and data, for example to expose CICS assets RESTfully, or to serve modern Web 2.0 AJAX front-ends for your CICS programs. Dynamic Scripting applications simply consist of scripts and configuration files on the zFS file system, so they can be developed with the tooling of your choice.
- Applications running on the Feature Pack can tightly integrate with existing CICS applications and data, including COBOL assets. They inherit the strengths of CICS and z/OS, including their Quality of Service characteristics.
- Project Zero, per the Project Zero Web site “began life as an incubator project to explore a new idea” … “of a development and runtime environment that could revolutionize creation of dynamic web applications – providing a powerful development and execution platform for modern Web applications while at the same time having the overall experience of being radically simple” . Users of Project Zero technology include the CICS Dynamic Scripting Feature Pack, the WebSphere Application Server Dynamic Scripting Feature Pack, and WebSphere sMash.
- **WebSphere sMash** – is an implementation of the Project Zero technology. A fully licensed retail version of IBM WebSphere sMash is available for production use. An IBM WebSphere sMash Developers Edition is available for free when used for development and limited deployment (see license details).

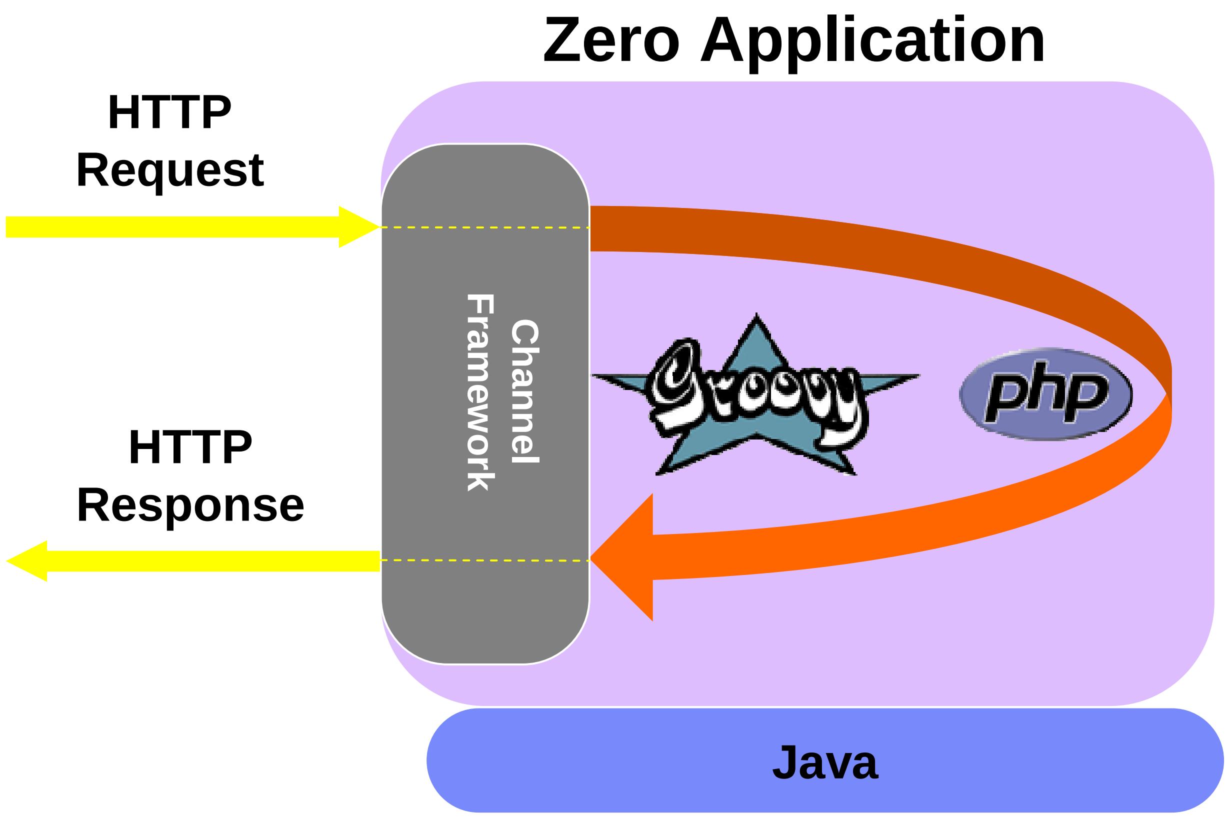
Installation

- Check prereqs – CICS TS V4.1 APARS, z/OS, Java, DB2
- Jar –xvf filename.zip
  - Creates cics and zero directories
- Export
  - ZERO_HOME=/install_directory/zero
  - PATH=$ZERO_HOME:$PATH
  - STEPLIB=CICS.TS.SDFHEXCI:CICS.TS.SDFHLOAD
  - JAVA_HOME=/java/java60_bit31_sr8/J6.0
- Create LOADLIB, add to DFHRPL
- Install
- Set permissions
- Configuration files are in UTF-8 encoding
- Well documented in the InfoCenter

Notes:

- Installation is well-documented in the CICS InfoCenter.
- The installation is straight forward with
  - Expanding a .zip file
  - Exporting environment variables
  - Creating a loadlib and adding it to your DFHRPL
  - Configuring your region for Java
  - And changing permissions on files
- Be sure to check that you have the prerequisite maintenance added to your system
- Additionally, be aware that the configuration files (and most all files used with Dynamic Scripting) are expected to be in the ISO-8859-1 encoding
- The CICS InfoCenter has information on options available to edit these files

Project Zero Environment (e.g. WebSphere sMash)



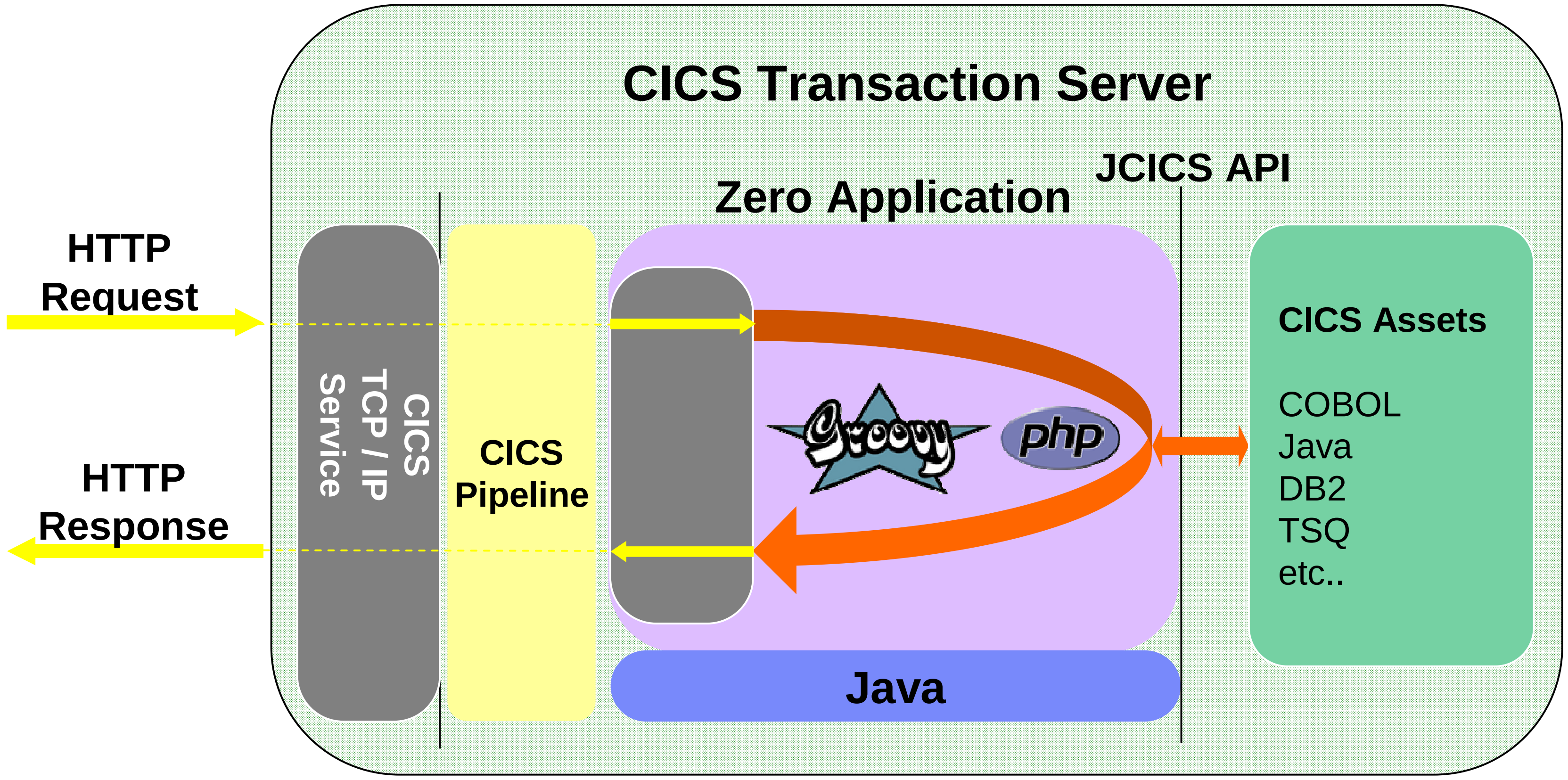
Notes:

- From a Project Zero developer's perspective, the Application is the server. This is in contrast to the normal thinking where you have a server and run multiple applications under that server. When a Project Zero application is started, it has its own port and takes care of all HTTP and database interactions. It is like the infrastructure is an extension of the application (versus the normal thinking of the application being an extension of the infrastructure).

- How this is physically applied is that you 'create' an application, add application code, then 'start' the application. That's it. The application listens on a specified port, and responds to HTTP requests as appropriate. The capabilities like listening/responding to HTTP, interacting with a database, using Dojo, etc are added to the application by adding 'dependencies' to the ivy.config file in the application’s /config directory (more on this later).

- At a lower physical level, each application runs in its own JVM.

Project Zero Environment (in CICS)



Notes:

- When applied to CICS, HTTP requests go through CICS (so CICS can apply security) and are then passed to the Dynamic Scripting Application. Each Dynamic Scripting application in CICS runs in its own JVMServer. The JVMServer in CICS is a multi-threaded JVM. Any JVMs is multi-threaded, however in CICS, each of the threads used for application code are associated with a T8 TCB (the new TCB type for CICS TS V4.1). The reason for the T8 TCBs is that although you can create new threads in a JVM, CICS won't be aware of them unless they are mapped to T8 TCBs. A T8 TCB is needed for application code on the thread to be able to interact with CICS. So, if CICS is creating threads in the JVM, T8 TCBs will be mapped to the threads and code running on those threads can interact with CICS. If an application programmer does a Thread.create() (or similar function), then the thread won't be mapped to a T8 TCB, CICS will be unaware of the thread, and code running on the thread cannot interact with CICS. (Bottom Line: application programmers are discouraged from creating their own threads).

- A Dynamic Scripting application can have "hundreds" of concurrent requests executing in a single JVMServer. Each of these threads would be a concurrent path through the application.

- The JVMServer resource has a THREADLIMIT() parameter where you can specify the max number of threads (T8 TCBs) that can be allocated to the JVMServer. The ThreadLimit on a JVMServer can be from 1-256 with the default being 15. There can be a maximum of 1024 threads for a CICS region. The number of JVMServers will also be influenced by the size of the JVM implemented by the JVMServer resource. These threads are for concurrent application usage. This means that a single JVMServer with 256 threads, depending on the request arrival rate, could be able to handle multiple thousands of users.

Command Line Interface (CLI) (from USS)

- **Commands**
  - zero create <application_name>
    - Create the directory structure
  - zero start
  - zero resolve
  - Many more
- ‘**zero**’ commands from the command line
  - Run in a JVMServer
  - Cause CICS resources to be dynamically created
- Available ‘**zero**’ commands
  - When in an ‘application’ directory, go against your application
    - e.g. zero start
  - When not in an application directory
    - e.g. zero create

Looks like any other Project Zero environment

Notes:

- The interface to the Project Zero technology implemented in CICS Dynamic Scripting is the same as with any other Project Zero implementation; a command line interface (CLI).
- The exceptions to the Project Zero command line interface is that the WebSphere sMash Eclipse plug-in allows you to start and stop your application by selecting options from the context menu presented by right-clicking on your WebSphere sMash project. When using the AppBuilder you can access the command line interface through the Web browser interface.
- The available ‘zero’ commands are different as to whether you are in an ‘application’ directory or not. If you are in your application directory you can do commands such as ‘zero start’, ‘zero stop’, and ‘zero resolve’ which go against your application. When outside your application’s directory you can use commands like ‘zero create’ to create a new application.
- The CLI (command line interface) communicates to CICS using EXCI so the resource definitions supplied with Dynamic Scripting include a CONNECTION definition.

CICS Resources

- CICS resources are **dynamically created** to run your application
- All CICS resource names constructed from a fixed prefix plus a counter
  - **JVMSERVER**: JCnnnnnn
  - **PIPELINE**: PInnnnnn
  - **TCPIPSERVICE**: TPnnnnnn and TSnnnnnn
  - **URIMAP**: UPnnnnnn and USnnnnnn
- Id based on a counter maintained in a CICS TSQ “ZEROJID”
  - Same id used for all resources installed for an application
- User **can override defaults** for resource options by specifying values in zerocics.config
  - For example, number of threads and JVM size
  - **Some options cannot be overriden** by zerocics.config
    - JVMSERVER: JVMPROFILE
    - PIPELINE: CONFIGFILE
    - TCPIPSERVICE: PORTNUMBER and PORTADDRESS
      - *Values specified for application in zero.config are used*
    - URIMAP: USAGE(PIPELINE), SCHEME, PIPELINE and TCPIPSERVICE

Notes:

- Any requested ‘zero’ commands are run in a JVMServer.
- In the case of a ‘zero start’, multiple CICS resource definitions and configuration files are dynamically created.
- The resource definition names contain two letters and a 6 digit number. The number is maintained in a TSQ named ZEROJID. Since there may be multiple sets of CICS resources at the same time (for different zero commands), each set must be unique.
- The port that your application will listen on is taken from a config/zero.config file in your application’s directory and placed in a TCPIPSERVICE definition. The TCPIPSERVICE resource definition will start with the letters ‘TP’ or ‘TS’ depending on whether you are using SSL or not.
- The JVMServer characteristics will be taken from a configuration from your application’s config/zerocics.config file. Some JVM characteristics can be specified, but some options cannot. The JVMServer name will start with the letters ‘JC’.
- A URIMAP definition is created (which starts with either ‘UP’ or US’ depending on whether you are using SSL or not), to direct all incoming HTTP activity from the port specified in the TCPIPService definition to your JVMServer.
- The URIMAP definition does not point directly to your JVMServer, but points to a PIPELINE definition (starts with ‘PI’) whose corresponding pipeline configuration file specifies a handler that directs all activity to your JVMServer.

Other dynamically created resources

- **JVMSERVER profile** (JVMProfile file)
  - Created in directory identified by JVMPROFILEDIR system initialization parameter
  - Built up from:
    - Command line generated by zero script's (class path, java cmd line options)
    - Information in zerocics.config JVMPROFILE section
    - sMash applications resolved.properties file (LIBPATH)
    - Dynamically created Java system properties defined for use by Java code (z/OS UNIX code page, zero cmd aguments)
    - WORK_DIR set to APP_HOME
- **Pipeline configuration file**
  - Created in APP_HOME/.zero/private/cics
  - Created from hard-coded template plus applications JVMSERVER id
    - Pipeline handler set to "zero.cics.CicsAdapter"

Notes:

- When CICS starts the JVM running under the control of your JVMSERVER resource, CICS needs to know the JVM's characteristics. As with other JVMs in CICS, the characteristics of the JVM are specified in a JVMProfile file.

- This JVMProfile file is dynamically created for you and placed in the JVMPROFILEDIR specified in the SIT (system initialization) parameters.

- The contents of JVMProfile file are built from the
  - Command line generated by zero script's (class path, java cmd line options)
  - Information in zerocics.config JVMPROFILE section
  - The zero application's resolved.properties file (LIBPATH)
  - Dynamically created Java system properties defined for use by Java code (z/OS UNIX code page, zero cmd aguments)
  - WORK_DIR set to APP_HOME

- A pipeline configuration file used by the PIPELINE resource is placed in your application's .zero/private/cics directory.

- The pipeline configuration file contains the JVMSERVER id and a handler. The name of the handler is zero.cics.CicsAdapter.

- Note that all resource definitions and configuration files that are dynamically created are also dynamically removed when the JVMServer is shut down. The exception is that you can set an environment variable to tell CICS to leave some of the configuration files and file containing details about the actions that were taken. The name of the environment variable is CICS_DEBUG=ON

Configuration Files

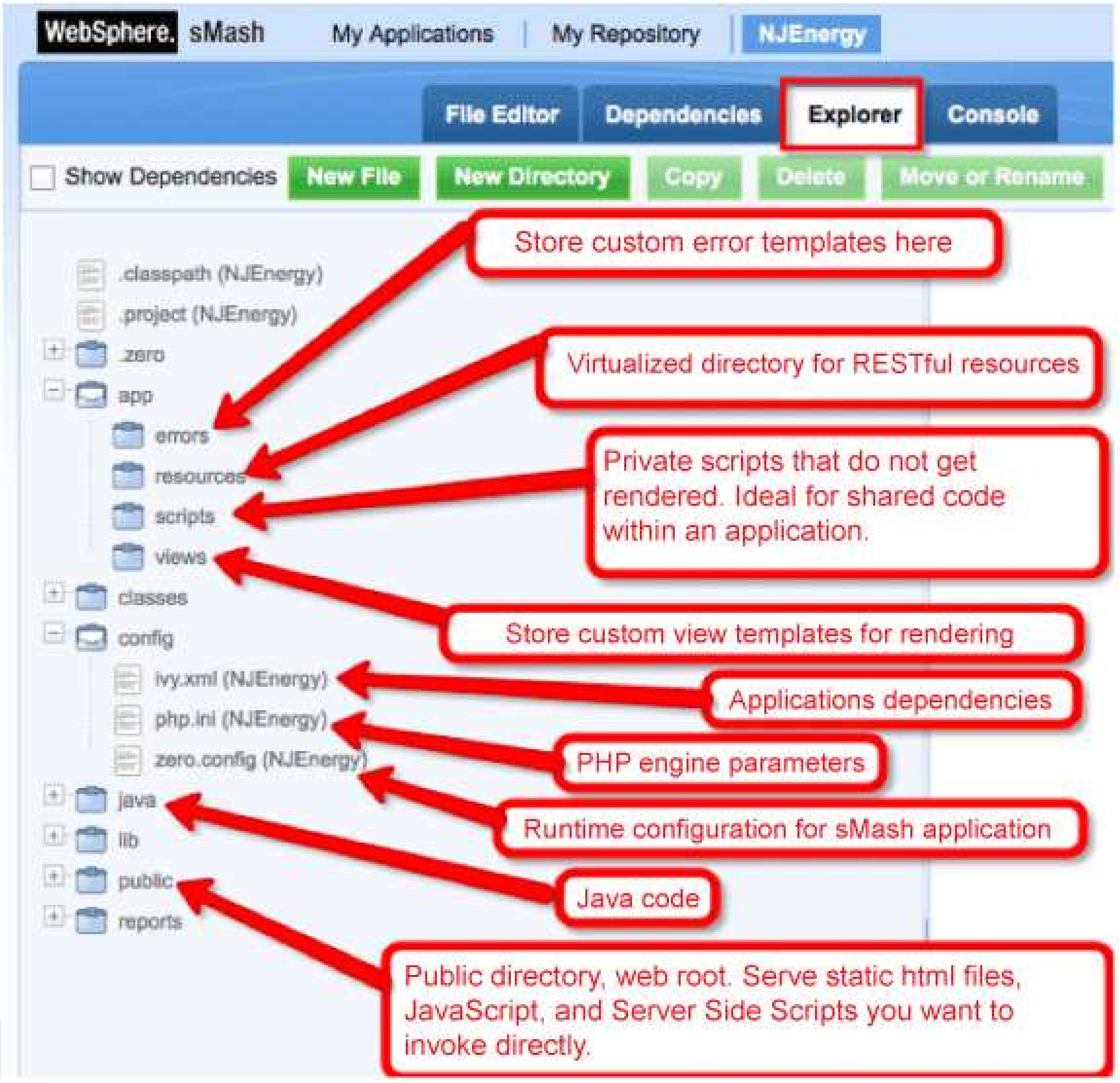
- Two configuration files – **not to be confused**.
- **zero.config:** contains general application configuration settings
  - **Every module** has a zero.config file in APP_HOME/config
  - Part of zero programming model – not specific to CICS Dynamic Scripting, also used in sMash
  - JSON format
  - Example settings: application port numbers, custom event handlers, app-specific JVM options, custom settings to be added to global context config zone
  - Settings are inherited from dependencies' zero.config (and can be overridden)
- **zerocics.config**: defines CICS Dynamic Scripting's relationship with CICS region
  - **The CICS Dynamic Scripting installation** has a zerocics.config in $ZERO_HOME/config
  - Specific to CICS Dynamic Scripting – never used in plain WebSphere sMash
  - Example settings: which CICS region to use, which CICS user IDs, CICS resource attributes
  - Can be overridden on a per-app basis, but settings are **not** merged or inherited from multiple locations. One of two possible files is used exclusively:
    - $APP_HOME/config/zerocics.config is used if it is present
    - Otherwise, $ZERO_HOME/config/zerocics.config is used

Notes:

- There are two main configuration files that you will probably need to customize.
- The zero.config file occurs in every Project Zero module (application) and is in the module's config directory. This configuration file is not specific to CICS, but occurs in every module in every implementation of the Project Zero technology.
- The slide lists examples of the type of information contained in the zero.config file.
- The zero.config file contains the port your application will listen on. When your application is created this port is set to 8080 so it is likely that you will want to change the port. The port specified in the zero.config file will be used in the TCPIPSERVICE definition CICS dynamically creates for your application.
- There is also a zerocics.config file. There is a zerocics.config file in the CICS Dynamic Scripting installation's config directory. The contents of the installation config/zerocics.config file can be overridden by adding a config/zerocics.config file to your application's directory.
- The slide contains examples of what goes in this file, but include the connection over which the CLI communicates to your CICS region, defaults for USERID and the tranid used for zero requests, plus any overrides for resources that are dynamically created.

Project Zero Application

- A 'well-known' **directory structure**
- Base directory for HTML pages is public (or public/secure)
  - HTML, CSS, JavaScript, Graphics
- /app/resources for RESTful resources
- /app/views for Groovy Templates



Store custom error templates here

Virtualized directory for RESTful resources

Private scripts that do not get rendered. Ideal for shared code within an application.

Store custom view templates for rendering

Applications dependencies

PHP engine parameters

Runtime configuration for sMash application

Java code

Public directory, web root. Serve static html files, JavaScript, and Server Side Scripts you want to invoke directly.

Notes:

- Each Dynamic Scripting application has a standard (‘well-known’) directory structure. There are specific directories available for specific types of artifacts.   For example, the default location for HTML page is in your application’s ‘public’ directory. All of the directories (standard or optional) are documented in the Project Zero documentation.

- Project Zero applications enjoy a type of inheritance model. You could have base application A and specify that application B has a 'dependency' of application A. Application B would then inherit all of application A's functionality. Although in this case application A's artifacts wouldn't physically reside in application B's directory structure, for all practical purposes, application A and B are 'virtually' a single application. When displaying ‘virtualized’ directories for application B, application A and B's artifacts would be displayed as if they were physically a single directory structure, when in reality, their artifacts are not physically in the same directory structure.

- All applications are also “modules”. The above paragraph talks about a dependency on an application, but you would specify a dependency in Application B for module A.

- Dependencies are also for HTTP, database interactions, Dojo support, etc. The application's dependencies are specified in the ivy.config file in the application’s config directory. So if you want database support in your application B, you add that dependency to your application B's ivy.config. If you want Dojo support in your application B, you add that dependency to your application B's ivy.config file.

- We will talk more on dependencies, modules, and virtualized directories later in the presentation.

Application Development

- **z/OS**
  - Command Line Interface
  - Your favorite editor (vi, ISPF)
  - RDz (Rational Developer for System z)
- **Workstation** (export, import to z/OS)
  - Command Line Interface (plus your favorite editor)
  - AppBuilder (Web browser interface)
  - Eclipse Plugins (base Eclipse or RDz)
    - PDT – for PHP editor and debugger
    - Groovy – for Groovy editor and debugger
    - WebSphere sMash
      - *Create/start/stop Apps*
      - *Create/edit artifacts*
      - *Test/export/import*

Watch for encoding issues while editing files. See InfoCenter for details

When developing on the workstation, you can create a modulegroup and associate it with your application so that during development you only add items CICS allows.

Notes:

- You can develop, debug, and test your applications in a variety of locations using a variety of development environments.
- You can use text oriented editors such as ISPF, vi, or Notepad.
- Eclipse plug-ins are available for editing and debugging Java, PHP, and Groovy code (see the CICS InfoCenter).
- You can use RDz (Rational Developer for System z) or the Eclipse Target Management plug-in to edit files on your workstation that reside on z/OS USS (UNIX System Services). See the CICS InfoCenter for more details and a hyperlink to the plug-in.
- In addition to an Eclipse environment or just using text editors, the AppBuilder is also available for developing on a workstation. The AppBuilder has a Web browser interface so you can use a Web browser to access a different machine that is running the AppBuilder code. The AppBuilder is not currently supported to run under CICS.
- When developing on a workstation, you can create a modulegroup and associate it with your application. This ensures that during development you only add dependencies CICS allows. See the CICS InfoCenter on how to specify the modulegroup.
- The CICS InfoCenter has a procedure for migrating a Dynamic Scripting application developed on your workstation to a CICS Dynamic Scripting environment.
- Be careful editing files as most files are in UTF-8.

Basic Application Concepts

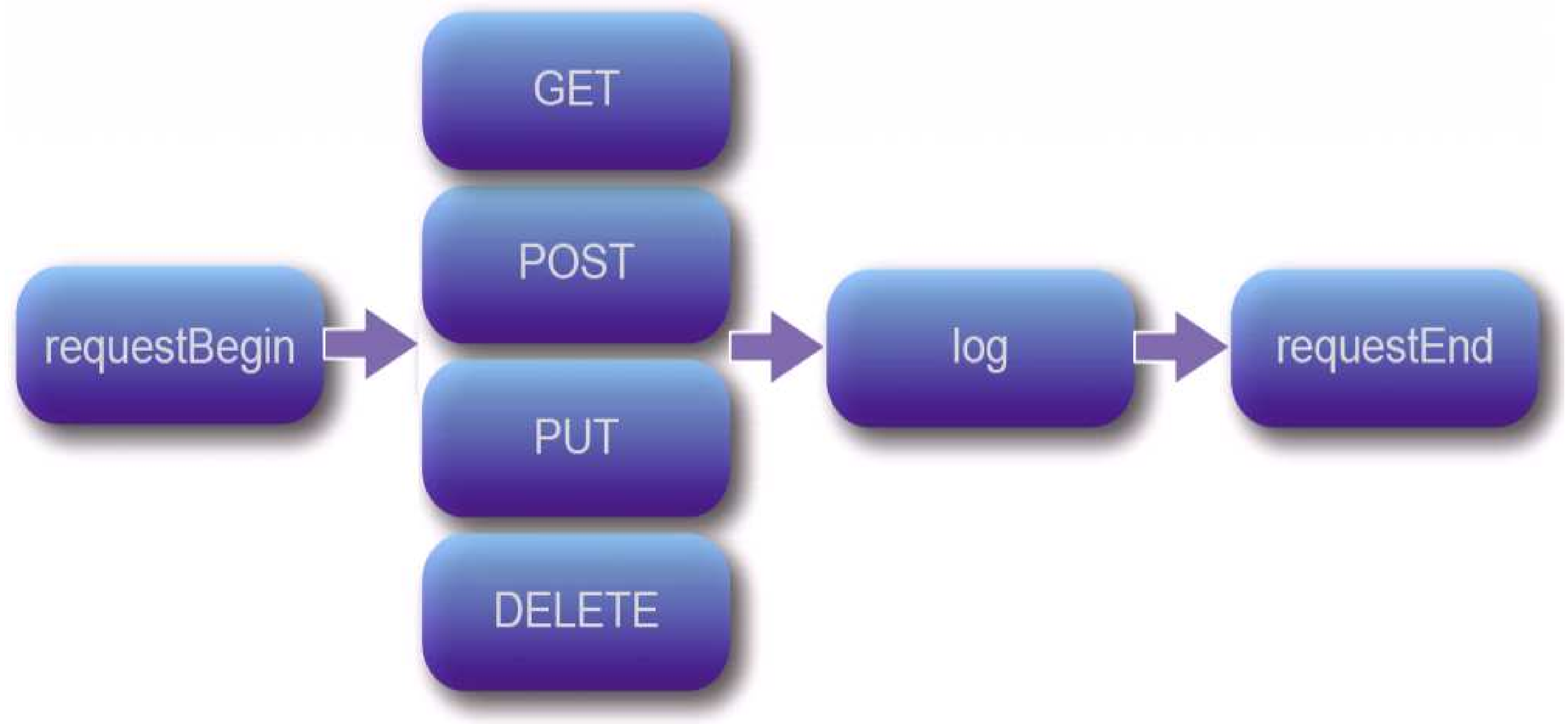
- **Command-line interface (CLI) from z/OS USS**
  - Previously discussed
- **Configuration**
  - zero.config and zerocics.config previously discussed
- **Zero Programming Model**
  - Events
  - Global Context
  - Notes on PHP Support
  - Zero modules
  - Resolving applications (dependency management)
  - Virtual Directories
  - ZRM (Zero Resource Management)
  - REST support
- **How to interact with CICS**

Notes:

- While developing your CICS Dynamic Scripting application, there are certain concepts you will need to understand.
- You interact with your application for administrative purposes from a USS (UNIX System Services) command line. You will need to have a basic understanding of the available ‘zero’ commands. These commands allow you to create an application, start the application, stop the application, resolve application dependencies, and much more.
- The zero.config and zerocics.config were discussed previously, but you will need a basic understanding of the items in these configurations files that affect your environment, for example the port your application will listen on is set in the zero.config file.
- From a programming perspective you will need a basic understanding of the facilities that are available to your application:
  - Events – your code, usually referred to as a handler, handles events in the Dynamic Scripting environment
  - Global Context – can be accessed to find out information about your environment or temporarily store items
  - PHP support – you can include PHP scripts in a Dynamic Scripting application
  - Zero modules – various features available to your application are supplied in Zero modules
  - Resolving dependencies – to include a feature, you specify that feature as a dependency
  - Virtualized Directories – a way to look at your application’s resources and all the resources it inherits
  - Zero Resource Management (ZRM) – a way to work with data in a Zero environment
  - REST support – Dynamic Scripting includes support for various aspects of REST
- You will also need a basic understanding of how to interact with your CICS resources using the JCICS API.

Events

- **All behavior in the system is modelled as a set of events**
  - Applications are built by handling these events and providing desired behavior



The zero programming model is single-threaded per request: for a given request, events are fired sequentially on the same thread.

Notes

- **Main Point:** a Zero application is an event-based system.
- All of the key behavior of the system is exposed to the application as a set of events, with the appropriate event state. Application developers mainly provide a set of handlers that hook into the well-known system events to achieve application behavior needed. The standard sMash events are meaningful activities of interest to applications.
- Common events that most application developers use, like providing a response to a GET request to a URI or when an application is started, are provided. Event handlers are stateless blocks of function that handle the events. Events are identified by an event name, such as GET or LOG. Event handlers indicate an interest in handling a particular event under a particular condition.
- Events in the Zero platform are ways to orchestrate behavior in the form of loosely-coupled event handlers. Zero "fires" a fixed set of events for HTTP request processing and stages of the application lifecycle; developers may add other event types. The application programming model is event based, so an event handler can be written in Groovy, PHP or Java, running in a browser or on the server side.
- "Firing" an event, which equates to an API invocation, causes Zero to invoke the associated handlers through an EventDispatcher. The set of associated handlers is determined by two mechanisms:
  - Explicit registration: Evaluate registration rules.
  - Implicit registration: Identify scripts as handlers through convention.

The Global Context

- It’s a map of data
- Language-agnostic
- Used for passing data between events, for storing application state
- Zones define the lifetime and visibility of the data

Zone	Scope/Visibility	
Event	All handlers for a single event	Non-persistent
Request	All handlers along the path of a single request	
Tmp	All requests for all users of the application	
Config	All requests all users	
User	All request for a particular user (HTTP Session equivalent)	Persistent
App	All requests for all users of the application	
Storage	All requests for all users of the application	

Notes:

- The global context is areas where your programs:
  - can access information about the current environment
  - Store/access information that is shared between all requests
  - Store/access information that is private to a request
  - Store/access information that persists between requests
  - Store/access information that exists only during the request

- The Project Zero documentation contains a list of all the zones and how they can be used.

PHP (PHP: Hypertext Preprocessor)

- **Can use PHP with CICS Dynamic Scripting**
- PHP is a dynamic scripting language typically used to implement Web sites (server side).
- More than 3 million developers worldwide (source: Gartner 12/2007)
- Predicted to grow to 5.5M developers, (60% corporate) by 2013 (source: Gartner 12/2007)
- 3rd most popular language (after Java/C) (source: TIOBE Programming Community Index (March 2010))
- PHP present on 20M+ Web domains (34% of internet) (source: www.netcraft.com)
- Efficient syntax and library have evolved in open source
  - Community driven to get more done in less time
  - Impressive results with little code
  - Extensive library support
  - Language suited to rapid incremental prototyping
- Influenced by: C, Perl, Java, C++, Tcl, Others

Notes:

- PHP is a dynamic scripting language typically used to implement Web sites (server side)
- More than 3 million developers worldwide (source: Gartner 12/2007)
- Predicted to grow to 5.5M developers, (60% corporate) by 2013 (source: Gartner 12/2007)
- 3rd most popular language (after Java/C/C++/VB) (source: TIOBE Programming Community Index (March 2010))
- PHP present on 20M+ web domains (34% of internet) (source: www.netcraft.com)
- Efficient syntax and library have evolved in open source
  - Community driven to get more done in less time
  - Impressive results with little code
  - Extensive library support
  - Language suited to rapid incremental prototyping
- PHP was invented in 1994 by Rasmus Lerdorf. Because of the Internet popularity, Rasmus wanted to come up with a simple language that could be used by anyone to quickly build Web pages. In 1994, Rasmus called his language Personal Home Page.
- PHP was refined by the open source community and evolved to what it is today.
- In about 1997, since the language was no longer just being used for personal home applications, the PHP acronym meaning was changed. It is now a recursive acronym and stands for PHP: Hyptertext Preprocessor.
- PHP is very popular and its use is expected to grow.

PHP Support

- The PHP engine is written in Java
  - Compiles PHP into Java bytecode to run on the Java Virtual Machine.
- **Does not support all PHP functions and extensions available for the original PHP engine from php.net, but can run many popular apps:**
  - phpBB, WordPress, SugarCRM…
  - See projectzero.org for limitations: http://www.projectzero.org/zero/monza/latest/docs/zero.devguide.doc/zero.php/Core.html
- **PHP / Java bridge allows PHP code to call Java APIs directly:**

```
<?php
java_import('java.util.HashMap');

$map = new HashMap;
$map->put('data', array(1,2,3,4,5));
var_dump($map->get('data'));
```

Import Java classes into PHP code

Use them just like PHP classes

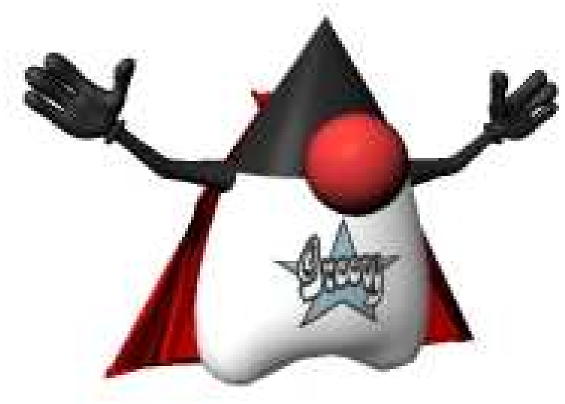
- **More information:** http://projectzero.org/php

Notes:

- The PHP interpreter used with CICS Dynamic Scripting runs on top of Java. The PHP code is compiled, at run time, into Java bytecode.
- There are multiple business applications that have been written in PHP and can run in the PHP support provided by CICS Dynamic Scripting.
- There is a ‘bridge’ from PHP to Java, so from PHP you can utilize most Java classes.
- The slide shows the use of the HashMap Java class. Note that a new instance of a HashMap is created that will be referenced by the $map variable. The put() and get() methods access the data stored in a HashMap.

Groovy

- **Dynamic,** object-oriented scripting language
- Power features inspired by Python, Ruby, and Smalltalk
- Provides scripting language to Java developers with **almost-zero learning curve**
- **Easy to read** and maintain
- Integrates with all existing Java classes and libraries
- **Compiles into Java bytecode**
- **RESTful default methods**
  - def onList()
  - def onCreate()
  - def onRetrieve()
  - def onUpdate()
  - def onDelete()
- Written by James Strachan and Bob McWhirter – August 2003



Notes:

- **Groovy –**

- Groovy was written in 2003 by James Strachan and Bob McWhirter.

- Groovy 1.0, released on January 2, 2007, is a dynamic scripting language. Wikipedia defines Groovy as “an object-oriented programming language for the Java platform. It is a dynamic language with features similar to those of Python, Ruby, Perl, and Smalltalk. It can be used as a scripting language for the Java Platform.”.

- Wikipedia goes on to say that “Groovy uses Java-like bracket syntax. It is dynamically compiled to Java Virtual Machine bytecode and interoperates with other java code and libraries. Most java code is also syntactically valid Groovy.”.

- Groovy contains ‘well-known’ methods that are convenient to use for RESTful requests.

- Groovy also compiles into Java bytecode.

Zero Modules

- **All applications are "<u>modules</u>"**
- **Modules declare dependencies on other modules in <u>config/ivy.xml</u>:**

```
<dependencies>
  <dependency org="zero" name="zero.cics.core" rev="[1.0.0.0, 2.0.0.0["/>
  <dependency org="zero" name="zero.data" rev="[1.0.0.0, 2.0.0.0["/>
  <dependency org="zero" name="zero.mail" rev="[1.0.0.0, 2.0.0.0["/>
</dependencies>
```

- **Modules <u>inherit</u> all assets (scripts, static files, java classes) from their deps**

- **In Dynamic Scripting, all applications depend at least on zero.cics.core**
  - Provides the core CICS integration functionality
  - Itself depends on zero.core, therefore pulls in the core standard zero functionality.

→ **Modules are not just for user apps:** core functionality of zero and CICS Dynamic Scripting is implemented in zero modules

Notes:

- All apps are re-usable modules by default.
- Dependency management is done using Apache Ivy via the ivy.xml configuration file.
- ivy.xml defines the name and version of the current module, as well as any dependencies the module has. Version ranges can be enforced on dependencies.
- If a module has a dependency, then:
  - Any scripts in the dependency are accessible from the current module
  - Any Java classes / libraries from the dependency are on the CLASSPATH
  - Any static files from dependencies (e.g. images or scripts) are accessible when accessing the app over HTTP
  - This relies on the concept of virtualized directories

Resolving Applications…

- **An application must be "resolved" before the it can be used. Resolving an app means:**
  - Locating its dependencies & determining exactly which versions to use.
  - Possibly retrieving them from a remote repository, if they are not found in the app's "workspace" or the CLI's local repository.

- **Two commands can resolve an app:**
  - **zero resolve**: attempts to locate the exact same versions of the dependencies that were used last time the module was resolved.
  - **zero update**: resolves the app against the latest suitable versions of the modules available in the local repository.

- **NB: These commands access a remote repository if no suitable version is found in the local repository.**

Notes:

- NB: “zero resolve” and “zero update” only contact the remote repositories if no suitable module is found locally.

- Once the app is resolved, the location of the dependencies is written to file:
  - $APP_HOME/.zero/private/resolved.properties

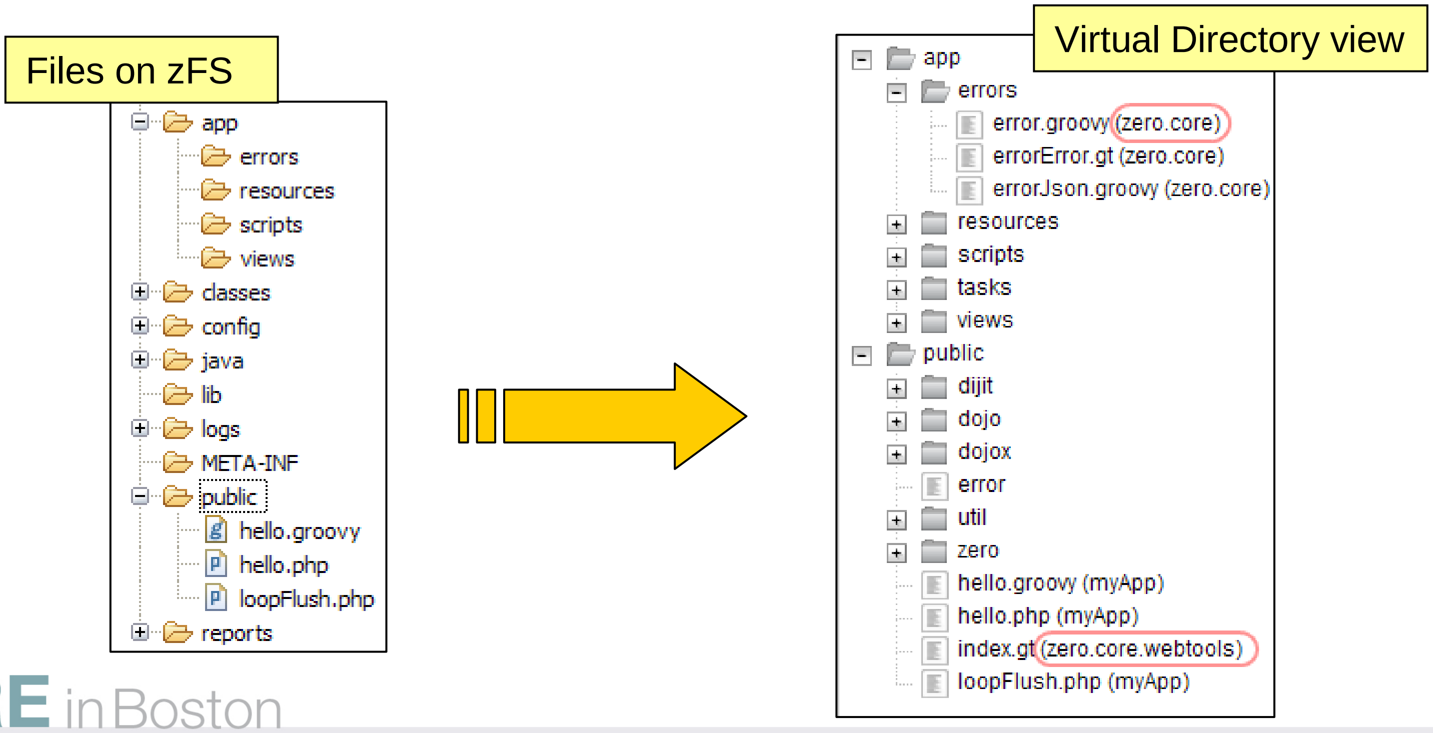
- This information is used to load the application’s classes.
  - Most dependencies are not part of the CLASSPATH when the JVM is started. They are added dynamically at runtime during application initialization.

- “resolve” and “update” look for modules in…
  - 1. The app’s “workspace”, i.e. the parent directory of the app.
    - Modules in the same workspace are referred to as “peers”
  - 2. The CLI’s local repository.
    - $ZERO_HOME/zero-repository/<*module_group_name*>
  - 3. Remote repositories.
    - The CLI’s current active module group defines which URIs will be searched. Ivy and Maven repositories are supported.
    - Users can add repo URIs to module groups and create new module groups
    - The default module group is called “stable”.

- More info on zero dependency & repository management:
    - http://www.projectzero.org/sMash/1.1.x/docs/zero.devguide.doc/zero.cli.tasks/DependencyManagement.html

Virtualized Directories

- **From the application developer's perspective, artifacts are "inherited" from dependencies.**
- **They are available through the concept of Virtualized Directories.**
  - The Virtualized Directory browser tool illustrates this. It can be added to any app by adding a dependency on the module zero.core.webtools.



Files on zFS

```
app
    errors
    resources
    scripts
    views
classes
config
java
lib
logs
META-INF
public
    hello.groovy
    hello.php
    loopFlush.php
reports
```

Virtual Directory view

```
app
    errors
        error.groovy (zero.core)
        errorError.gt (zero.core)
        errorJson.groovy (zero.core)
    resources
    scripts
    tasks
    views
public
    dijit
    dojo
    dojox
    error
    util
    zero
    hello.groovy (myApp)
    hello.php (myApp)
    index.gt (zero.core.webtools)
    loopFlush.php (myApp)
```

Notes:

- Each Dynamic Scripting application has a standard (‘well-known’) directory structure. There are specific directories available for specific types of artifacts.   For example, the default location for HTML page is in your application’s ‘public’ directory. All of the directories (standard or optional) are documented in the Project Zero documentation.

- Project Zero applications enjoy a type of inheritance model. You could have base application A and specify that application B has a 'dependency' of application A. Application B would then inherit all of application A's functionality. Although in this case application A's artifacts wouldn't physically reside in application B's directory structure, for all practical purposes, application A and B are 'virtually' a single application. When displaying ‘virtualized’ directories for application B, application A and B's artifacts would be displayed as if they were physically a single directory structure, when in reality, their artifacts are not physically in the same directory structure.

- All applications are also “modules”. The above paragraph talks about a dependency on an application, but you would specify a dependency in Application B for module A.

- Dependencies are also for HTTP, database interactions, Dojo support, etc. The application's dependencies are specified in the ivy.config file in the application’s config directory. So if you want database support in your application B, you add that dependency to your application B's ivy.config. If you want Dojo support in your application B, you add that dependency to your application B's ivy.config file.

REST - Representational State Transfer

- **Leverages HTTP protocol**
  - Nouns (URLs) indicate what is being worked on
  - Verbs (GET, PUT, POST, DELETE methods) indicate the action to be performed (List, Create, Read, Update, Delete)
- **Resource centric**
  - Similar in concept to hyperlinked data
- **Content negotiation**
  - REST does not restrict format of results
  - HTTP headers can be used to request format with no changes to URL
  - Popular formats of returned data are **XML and JSON**
- **Lightweight data transfer**
  - From Web browser or any HTTP client or server
- **More information:** http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Notes:

- REST (REpresentational State Transfer) is an architectural style that applies the approach we use to access Web pages to access our business data. Just like we use a URL to access the current state of a Web page, you use a URL to access the current state of business data. We can specify a specific Web page on a URL, we can also specify a specific account number on a URL.
- We normally need to perform LCRUD (List, Create, Read, Update, and Delete) functions on our business data. The HTTP ‘methods’ that flow with the request indicate the action to be performed on the data. Whereas we normally only use a GET or a POST method when accessing a Web page, for data, a GET method indicates a list or a read, DELETE for a delete, POST for an add, and a PUT for an update.
- REST results in very lightweight interactions with a minimal amount of characters transferred.
- The format of the returned data is not dictated, although most people use XML or JSON (JavaScript Object Notation.
- REST is documented in Roy Fielding’s year 2000 doctoral thesis. In his thesis, Fielding indicates that REST started in 1994 and was iteratively redefined. Since many people were not aware of REST, they think it is a followon to Web services, however Web services came after REST.
- For situations where you want interfaces documented with WSDL, transactionality, and more security options, Web services are great. Where you just need lightweight data access, REST is great.
- One of the primary uses of REST is for requests from Web browsers. JavaScript running in a Web browser can use AJAX (Asynchronous JavaScript and XML) to make RESTful requests to backend data and business logic systems such as CICS.
- ZRM (Zero Resource Model) discussed later can be used to very quickly expose a resource with a RESTful interface using single command called delegate.

REST and Project Zero

- **RESTful event handlers in Project Zero**
  - Each script in the **<apphome>/app/resources** directory is a resource handler
  - URL convention for interacting with resources:
    - /resources/<collectionName>[/<memberID>[/<pathInfo>]]
  - URI and HTTP method define the resource to access and the action to perform
  - Action can be taken on the entire collection, or a specified member of the collection
- **Example:**

URI	HTTP Method	Event Description	Resource Handler Function
http://example.com/resources**/people**	**GET**	List people	`onList()`
http://example.com/resources**/people**	**POST**	Create person	`onCreate()`
http://example.com/resources**/people/john**	**GET**	Retrieve person	`onRetrieve()`
http://example.com/resources**/people/john**	**PUT**	Update person	`onUpdate()`
http://example.com/resources**/people/john**	**DELETE**	Delete person	`onDelete()`

Notes:

- Let's take a look at how a RESTful service can be implemented using the Project Zero programming model.

- Each PHP or Groovy script placed in the /app/resources directory of a Project Zero application is automatically treated by the platform as a RESTful handler for a category of resources, or a "resource collection". The name of the script represents the name of the collection. This script contains the logic to execute when processing inbound HTTP requests for that resource, separated into functions with well-defined names. The function that is invoked depends on the URI and HTTP method of the inbound HTTP request.

- The URI pattern shown in the slide is a convention used to identify which collection to access based on the URI of an inbound HTTP request. If the URI contains just a collection name, the operation is targeted at the whole collection. If a member ID is specified in the URI after the collection name, the operation is targeted at an individual member of the resource collection. Optionally, additional information can be specified after the member ID.

- This table shows an example with a resource collection called "people". The URI column shows two different kind URIs that can be used to interact with instances of the resource: the collection URI, which ends with the collection name - in this case "people", and the member URI in which an identifier for an individual person is specified - in this case, the name "john". We can see how a request URI, combined with an HTTP method, triggers an event such as List, Create, Retrieve, Update or Delete. These events are sometimes referred to as "L-CRUD" events. By convention, the Project Zero platform searches for handlers for these events a script called "people.groovy" or "people.php" in the /app/resources directory. If this script provides an implementation of the function corresponding the the event, that function is invoked to handle the request.

- Therefore, you can develop a RESTful service simply by creating a single script and implementing the subset of L-CRUD functions that you need. The platform takes take care of mapping inbound requests to your logic, by following a set of RESTful conventions.

Zero Resource Model

- **Model application data**
  - Constrained set of APIs encourages a RESTful application architecture
  - Data model that maps well into Atom feeds and JSON formats
  - Robust framework for persistence, validation, and serialization
  - Application Database focus

Notes:

- **ZRM (Zero Resource Model)** – Per the Project Zero Web site, “The Zero Resource Model (ZRM) provides a simplified way to create a RESTful resource handler with a data store. Developers need provide only simple “model” definitions of resources; ZRM uses the model definitions to create the data store and support full create/read/update/delete semantics. In addition, ZRM supports a variety of content formats, including JSON and Atom Publishing Protocols.

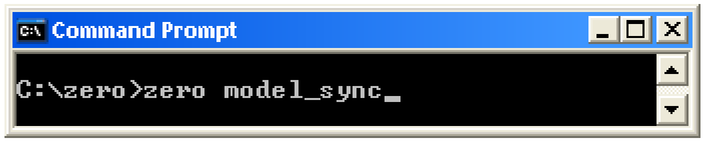
ZRM Development Life Cycle

app/models/employee.json

```
{
    "fields" : {
        "first_name": {"type":"string"},
        "last_name": {"type":"string"},
        "location": {"type":"string"}
    }
}
```

app/resources/employee.groovy

```
ZRM.delegate();
```



```
C:\zero>zero model_sync_
```

app/models/fixtures/initial_data.json

```
[
    {
        "type": "employee",
        "fields": {
                "first_name" :    "Alice",
                "last_name“  :    "Rogers",
                "location"   :    "Seattle"
        }
    },
    {
        "type": "employee",
        "fields": {
                "first_name" :    "Bill",
                "last_name"  :    "Stevens",
                "location"   :    "Seattle"
        }
    },
    {
        "type": “employee",
        "fields": {
                "first_name" :    "Cathy",
                "last_name"  :    "Tomlin",
                "location"   :    "Boston"
        }
    }
]
```

Notes:

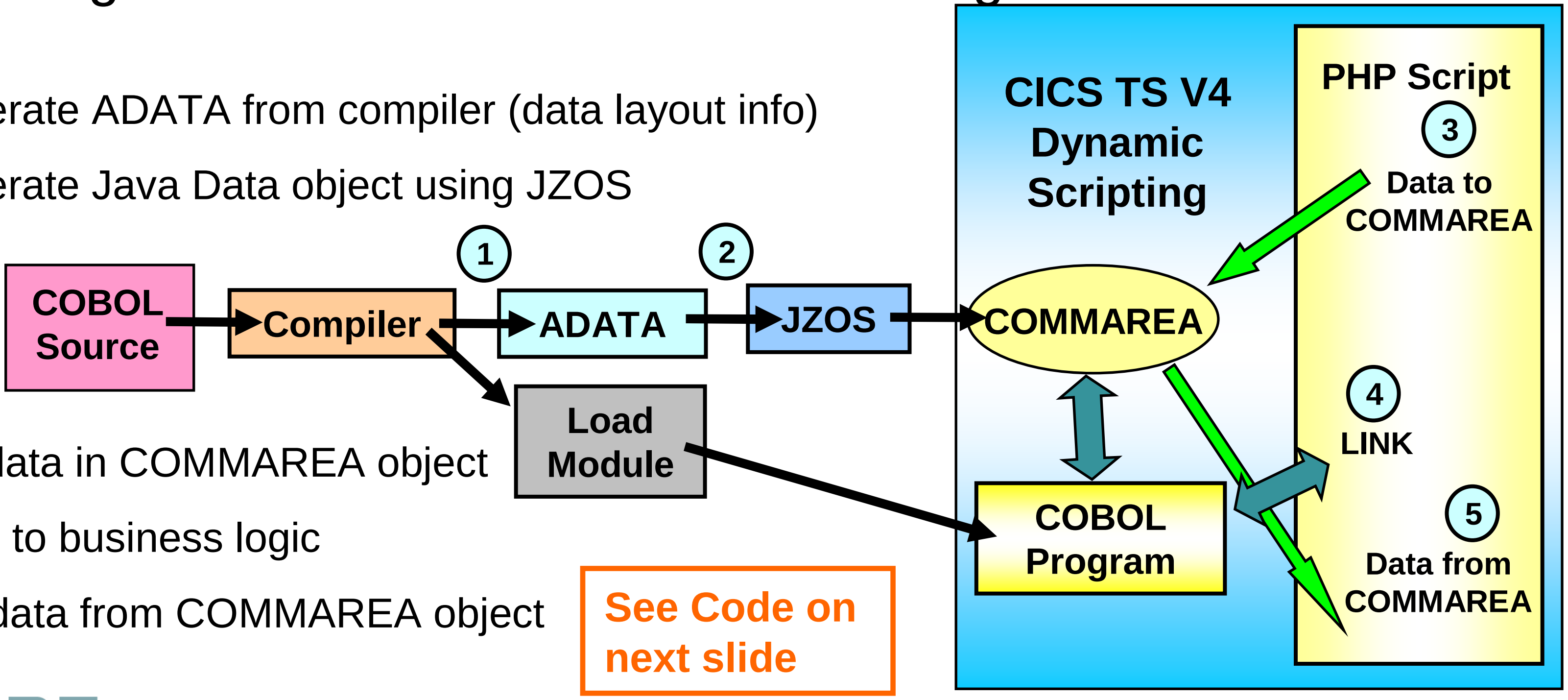
- This slide shows ZRM in action.
- In the top left corner of the slide is an illustration of how to define a simple data layout with three columns each of type string. This file is placed in the application's models directory.
- On the bottom left is the command that is used to create the data table.
- If you wanted to expose the data in the table with a RESTful interface you only need to add one line to a Groovy program (middle-left) in your resources directory.
- On the right is an illustration of how to load initial data into the data table.

Interacting with CICS

- Data passed to or from CICS is in byte arrays
- Can generate a Java data class with getters and setters plus a method to get and set the data as a byte array
  - Using JZOS classes supplied with Java on z/OS
  - Using RAD and CICS Java Data Bindings

1. Generate ADATA from compiler (data layout info)
2. Generate Java Data object using JZOS
3. Set data in COMMAREA object
4. LINK to business logic
5. Get data from COMMAREA object

See Code on next slide



Notes:

- Because the PHP interpreter used in CICS Dynamic Scripting is implemented in Java, there is a 'bridge' from PHP to Java. This allows you to access many Java functions, for example, some of the JCICS classes.
- This bridge also allows you to instantiate data objects that can be passed to the Java functions.
- In the case of invoking a CICS program using a COMMAREA interface, we need a way to construct the series of bytes that make up a COMMAREA. To do this, we can use classes from the JZOS product, or wizards in Rational Application Developer (RAD).
- JZOS is supplied with the System z Java implementations. The JZOS Java classes allow you to perform many z/OS specific functions such as reading a PDS or a VSAM file. One of the functions of JZOS is to generate a Java class that corresponds to a data structure in a high-level language such as COBOL.
- As illustrated on the slide, you can compile your COBOL program with the ADATA compiler option. The results of the ADATA compiler option can be input to JZOS classes that generate a Java equivalent of the data structure. There are getter and setter methods that correspond to the fields in the data structure plus a getBytes() method that allows you to get the series of bytes that correspond to the data structure.
- After the Java classes that correspond to the data structure that represents the COMMAREA are generated, you can instantiate the Java class in your PHP program, invoke setters on the Java object to provide data such as setEmployeeId(), then pass the data object to the JCICS LINK request.
- See the next slide for a coding example.

Interfacing with CICS

```
<?php
// Instantiate a COMMAREA representation
// The com.mycompany.EMPLOYEE_CommArea class created from
// a COBOL data layout using JZOS classes supplied with z/OS Java
$commArea = new Java('com.mycompany.EMPLOYEE_CommArea');
// Set some data in the commarea by calling method on the class
$commArea->setEmployeeNumber('115');
// Use the JCICS class to call a CICS program
$program = new Java('com.ibm.cics.server.Program');
$program->setName('EMPLOYEE');
try {
    $program->link($commArea->getBytes());
} catch (CICSException $e) {
    echo $e->getMessage();
    exit;
}
echo "Return value is " . $commArea->getReturnValue();
?>
```

Notes:

- For the code example on this slide, we would have compiled the target CICS program (EMPLOYEE in this case) with the ADATA compiler option. We would have used the ADATA information representing the COMMAREA of the EMPLOYEE program as input to the JZOS classes to generate a Java object that represents the COMMAREA (which would be called com.mycompany.EMPLOYEE_Commarea (or whatever name we wished to use)).

- In the code example we use a “new Java()” request to get an instance of the class that represents the EMPLOYEE program’s COMMAREA.. We then invoke methods on the class to set values (the example invokes the setEmployeeNumber() method).

- After data values are set in the object that represents the COMMAREA, we create a new Program object and use the setName() method to indicate the program we are referring to has a name of “EMPLOYEE” (because EMPLOYEE is the name of the target CICS program). We then invoke the link method of the CICS Program object, passing the byte array that represents the COMMAREA.

- In the code example, you can see that after the program invocation, we are accessing getters in the data object to obtain the information returned by the EMPLOYEE program in the COMMAREA.

- This slide illustrates a LINK to a program using a COMMAREA, but channels and containers may also be used, plus many other CICS API are supported.

- **JCICS JavaDoc:**
  - http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=/com.ibm.cics.ts.jcics.javadoc/com/ibm/cics/server/package-tree.html

Limitations

- **<u>Some restrictions</u> in Dynamic Scripting in CICS compared to WebSphere sMash**
  - Not all sMash modules are supported
  - Some config settings not supported
  - Automatic App recycling not supported
  - Some PHP extensions not supported
- **<u>Not all JCICS APIs</u> are supported**
  - e.g. XCTL, Web Send, 3270, APPC
- **Some minor differences in HTTP behavior between Dynamic Scripting in CICS and WebSphere sMash on other platforms**
- **Must serialize access to DB2 per application**
- **See CICS Dynamic Scripting in the CICS InfoCenter for more details**

Notes:

- There are some limitations and differences between CICS Dynamic Scripting and WebSphere sMash. These differences are listed in the CICS InfoCenter.
- Some of the limitations are listed here.
- **See the CICS InfoCenter for details on limitations**

Encoding

- **Other Encoding Considerations:**
  - All script files, configuration files etc… read by the zero platform must be UTF-8
  - All log files generated by the zero platform are UTF-8
  - Groovy introduces more convenience methods that use the default encoding
    - e.g. File.getText()
  - In the PHP language, there is no distinction between strings and byte arrays
    - String literals are arrays of UTF-8 bytes
    - Pay particular attention to encoding issues when passing PHP strings to Java methods over the PHP/Java bridge (see doc)

- **→ See CICS InfoCenter for more information:**

http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/topic/com.ibm.cics.ts.smash.doc/smash_dynamicscripting_encodingconsiderations.html

Notes:

- The Dynamic Scripting is basically a UTF-8 platform running on an EBCDIC system.

- The JVM running Dynamic Scripting must run in an EBCDIC JVM for the JCICS Java classes to work properly.

- Never rely on Java's default character set:
  - In Java, if no Charset is specified on a byte[] ↔ String conversion, a platform-specific default is used.
  - The default can be overridden with the file.encoding system property, but some JCICS library APIs require that the default is left unchanged.
  - Therefore, Dynamic Scripting apps run on JVMs that default to EBCDIC …
  - … but many 3rd Java libraries assume they are running on JVMs that default to ASCII

  - Consider the following call, which sends bytes over the network. If the recipient is expecting ISO8859-1 bytes, this call will work as expected on ASCII platforms, but not in CICS Dynamic Scripting:
    - connection.getOutputStream().write("HELLO".getBytes());
  - A portable equivalent would be:
    - connection.getOutputStream().write("HELLO".getBytes("ISO8859-1"));

- **See the CICS InfoCenter for details on encoding**

Trouble Shooting

- **PHP and Groovy editors and debuggers avaialble**

- **See Troubleshooting doc:**
  - **http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/topic/com.ibm.cics.ts.smash.doc/troubleshooting.html**

- **Examples:**
  - Pre-reqs missing
  - UNIX file permission issues
  - EXCI connectivity problems (CICS not up, incorrect STEPLIB environment variable, incorrect SVC number…)
  - App is missing dependency on a required module (zero.cics.core, zero.cics.php)
  - Attempts to use sMash features not yet supported in CICS Dynamic Scripting
  - App transferred from sMash to CICS with “package standalone” option
  - Zero programming model issues (not specific to CICS Dynamic Scripting)
    - See docs and forums on projectzero.org
  - JVMServer heap size configuration issues.
  - **Encoding issues**

Notes:

- The CICS InfoCenter has a good section on Trouble shooting.
- Additionally, there are PHP and Groovy editors and debuggers available.

- **See the CICS InfoCenter for details on trouble shooting**

Skill Set

- **Project Zero technology**
  - Command line interface (create/start/stop/sync)
  - Dependencies
  - Application directory structure
- **Optional:**
  - HTML, Cascading Style Sheets, JavaScript
  - Dojo
  - Eclipse (and PHP and Groovy plugins)
  - ZRM
  - WebSphere sMash features (e.g. security)
  - SQL, PHP, Java, Groovy, Groovy Template files
  - JCICS classes

Notes:

- The skill set you need to leverage the CICS Dynamic Scripting Feature Pack will vary greatly depending on the task you are try to accommodate.

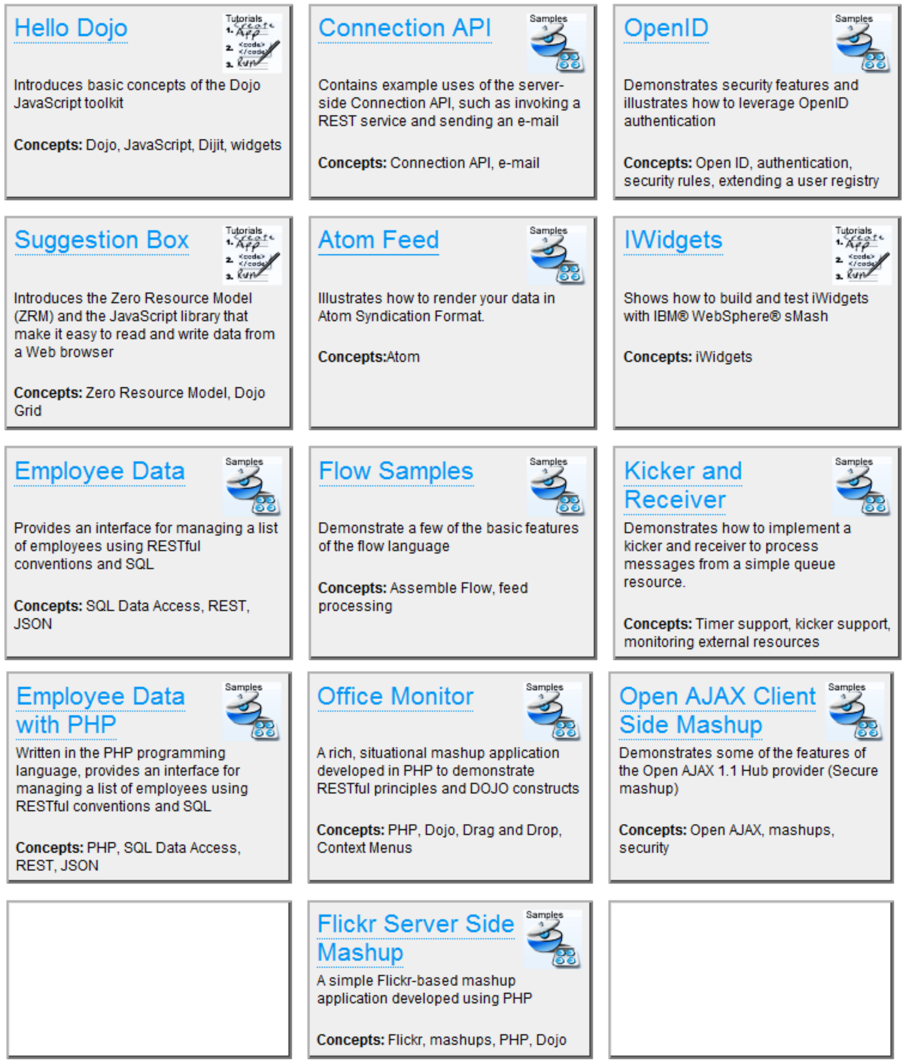
- You will find at least four primary sources of information when using the CICS Dynamic Scripting Feature Pack:
  - The CICS InfoCenter for issues around how the Project Zero technology is integrated into CICS and also the JCICS Java classes
  - The Project Zero Web site for information about the Project Zero technology
  - The WebSphere sMash Info Center
  - The Internet for information on PHP, Groovy, JavaScript, Dojo, etc

- You will need to have a familiarity with the ‘zero’ commands and using the CLI (Command Line Interface), even if you use the WebSphere sMash Eclipse plug-in for development. You can always search the Project Zero documentation for details on ‘zero’ commands, but you should at least know which command is needed for a specific purpose.

- You will also need to have a familiarity with zero concepts such as dependencies, ZRM, etc.

- Depending on the task at hand, you may also need to know:
  - HTML, JavaScript, Cascading Stylesheets, Dojo
  - SQL, PHP, Groovy, Groovy templates
  - Java
  - Security

Tutorials, Samples, and Demos



Hello Dojo (Tutorials)

Introduces basic concepts of the Dojo JavaScript toolkit

Concepts: Dojo, JavaScript, Dijit, widgets

Connection API (Samples)

Contains example uses of the server-side Connection API, such as invoking a REST service and sending an e-mail

Concepts: Connection API, e-mail

OpenID (Samples)

Demonstrates security features and illustrates how to leverage OpenID authentication

Concepts: Open ID, authentication, security rules, extending a user registry

Suggestion Box (Tutorials)

Introduces the Zero Resource Model (ZRM) and the JavaScript library that make it easy to read and write data from a Web browser

Concepts: Zero Resource Model, Dojo Grid

Atom Feed (Samples)

Illustrates how to render your data in Atom Syndication Format.

Concepts: Atom

IWidgets (Tutorials)

Shows how to build and test iWidgets with IBM® WebSphere® sMash

Concepts: iWidgets

Employee Data (Samples)

Provides an interface for managing a list of employees using RESTful conventions and SQL

Concepts: SQL Data Access, REST, JSON

Flow Samples (Samples)

Demonstrate a few of the basic features of the flow language

Concepts: Assemble Flow, feed processing

Kicker and Receiver (Samples)

Demonstrates how to implement a kicker and receiver to process messages from a simple queue resource.

Concepts: Timer support, kicker support, monitoring external resources

Employee Data with PHP (Samples)

Written in the PHP programming language, provides an interface for managing a list of employees using RESTful conventions and SQL

Concepts: PHP, SQL Data Access, REST, JSON

Office Monitor (Samples)

A rich, situational mashup application developed in PHP to demonstrate RESTful principles and DOJO constructs

Concepts: PHP, Dojo, Drag and Drop, Context Menus

Open AJAX Client Side Mashup (Samples)

Demonstrates some of the features of the Open AJAX 1.1 Hub provider (Secure mashup)

Concepts: Open AJAX, mashups, security

Flickr Server Side Mashup (Samples)

A simple Flickr-based mashup application developed using PHP

Concepts: Flickr, mashups, PHP, Dojo

Notes:

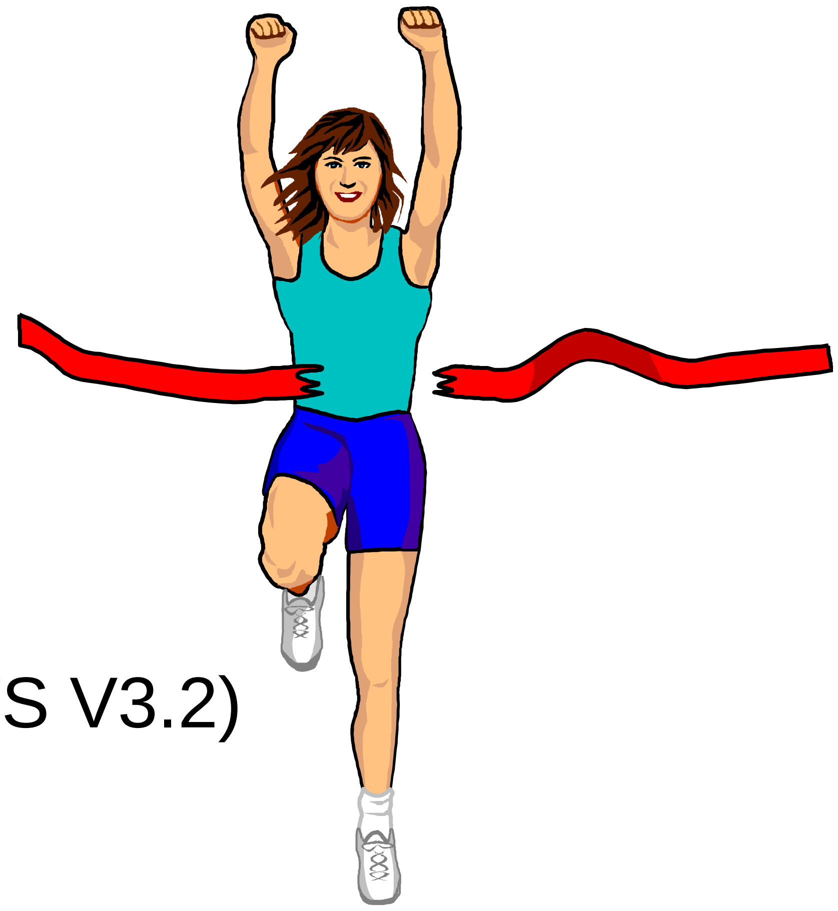
- An excellent way to grow your skills on CICS Dynamic Scripting is to look at the Tutorials, Samples, and Demos available on the Project Zero Web site.
- The CICS InfoCenter lists the Project Zero Tutorials, Samples, and Demos that work in CICS Dynamic Scripting.
- The CICS InfoCenter has directions on how to install Project Zero Demos in CICS Dynamic Scripting.

SupportPac CA1S

- Runs under CICS TS V3.2
  - Requires Java 5
- Provides PHP support in CICS for
  - Providing a RESTful interface
  - Prototyping Web Pages
- Can interact with CICS
  - LINK to COMMAREA-based CICS programs
  - Syncpoint and Rollback
- Can interact with DB2
  - Using PHP PDO

Summary

- CICS Dynamic Scripting Feature Pack
- Why
- What
- How
- When (now)
- What about CICS TS V3.x
  - SupportPac CA1S (CICS TS V3.2)



References

- **JCICS JavaDoc:**
  - http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=/com.ibm.cics.ts.jcics.javadoc/com/ibm/cics/server/package-tree.html
- **CICS InfoCenter:**
  - http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/topic/com.ibm.cics.ts.smash.doc/smash_overview.html
- **CICS on projectzero.org:**
  - http://projectzero.org/cics
- **ProjectZero forum:**
  - http://projectzero.org/forum
- **Tutorials:**
  - www.w3schools.com