CICS Dynamic Scripting:
Data Manipulation

Dennis Weiand
IBM

Tuesday, August, 10th, 6:00pm-7:00pm
Session #09608

Abstract

- This presentation discusses capabilities of the CICS Dynamic Scripting Feature Pack. You have seen overview presentations, but its time to start discussing some of the CICS Dynamic Scripting capabilities in detail.

  Data storage and access is an important part of any application, so this session will cover the Zero Resource Model (ZRM) to quickly define database tables and provide data access. Providing RESTful interactions in both XML and JSON will also be discussed.

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.

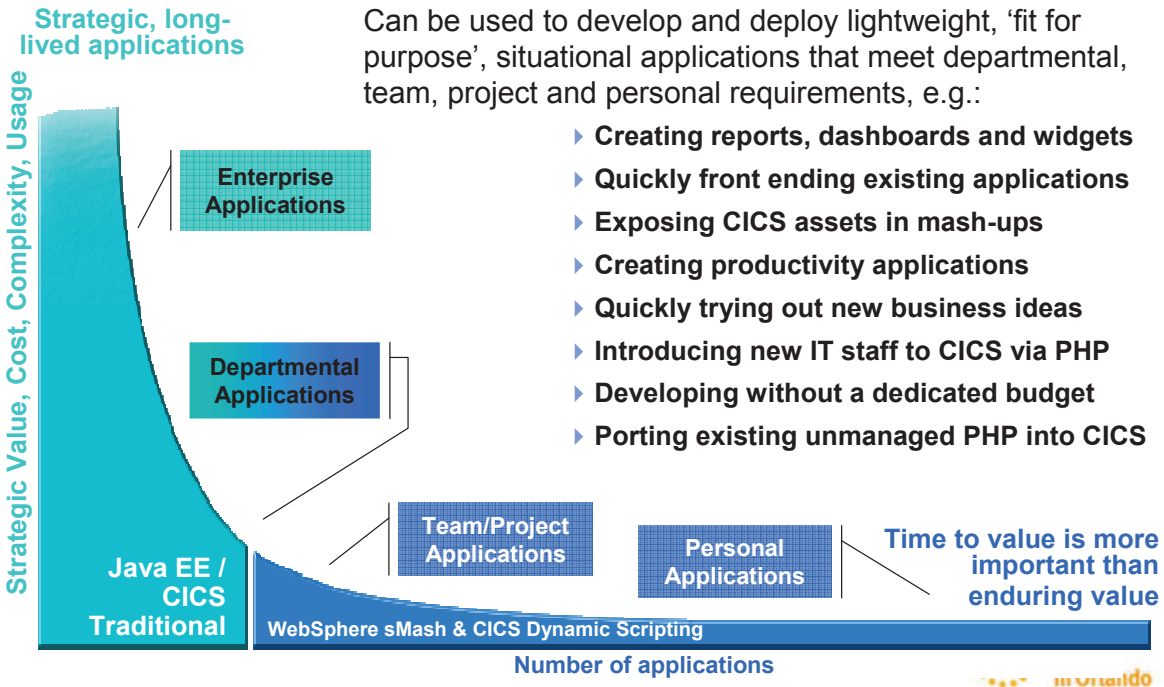
Agenda

- What is CICS Dynamic Scripting
- Review data-related concepts
- ZRM (Zero Resource Model)
  - Define/dump/load database tables
  - REST access and ZRM API access
- More database options
  - Configuration
  - Defining tables
  - Accessing table data
  - Implementing your own REST interfaces
- Formatting data (JSON, XML, ATOM)
- Accessing Data via the JCICS API

Notes:

- This presentation is intended to provide information on the data-related options available when writing CICS Dynamic Scripting Application. For data access, the easiest way will be to use ZRM (Zero Resource Model) which is a 'black box' implement to provide quick, easy database access. For more control you can configure the database details yourself, and can use SQL access. Since your application will be running in a CICS environment, you may wish to LINK to a CICS program that 'owns' the application data, or you can access CICS resource data direction (e.g. VSAM file).
- This Feature Pack became available on July 22, 2010 and is a no-charge feature of CICS TS V4.1
- As of the date on this presentation, the CICS Dynamic Scripting Feature Pack is only available to CICS TS V4.1 customers, there is a statement of direction for IBM to make the CICS Dynamic Scripting Feature Pack available under CICS TS V4.2.

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 or departmental applications we could use to enhance productivity. Because of the procedures and development techniques we use, we are often not able to address application requests for these 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 us 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.

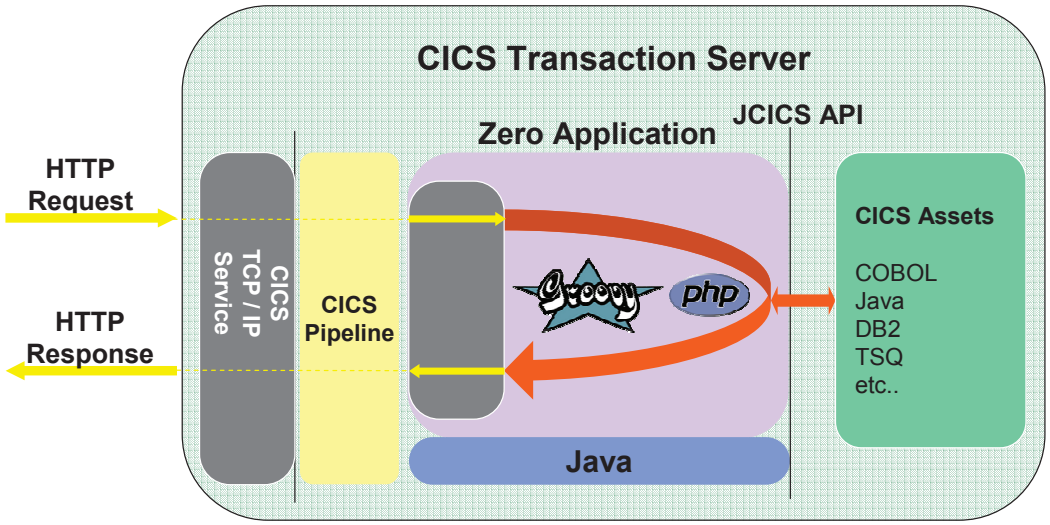
CICS Dynamic Scripting Feature Pack

- Technology from **Project Zero**, WebSphere sMash v1.1.1.3 (projectzero.org)
- Robust environment for **situational** reports, dashboards, and Web feeds
- Provides **PHP and Groovy** support in CICS – agile, productive environment
- Zero Resource Model (**ZRM**) with data managed by DB2 for z/OS
- Uses CICS TS V4.1 **JVMServer** Technology (statement of direction for CICS TS V4.2)
- Manageability, Scalability, and Security
- **Situational applications** - Quickly try business ideas
- Introduce **new staff** to CICS via PHP and Groovy
- 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 (currently a statement of direction for CICS TS V4.2)

Notes:

- CICS Dynamic Scripting is a Feature Pack for CICS TS 4.1.
- This feature pack embeds Zero's agile programming model into CICS on z/OS. This allows you to quickly construct Web applications, and 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).

Project Zero Environment (in CICS)



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.
- 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 JVM 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.

Concepts I will assume you know

- **Command-line interface (CLI) from z/OS USS**
  - Looks like any other 'project zero' environment
- **Configuration Files**
  - zero.config (application) and zerocics.config (CICS)
- **Upcoming slides on:**
  - **An application is a set of 'well-known' directories**
  - **Applications are coded in PHP, Groovy, and/or Java**
    - Use your favorite editor or development environment
  - **Applications are modules, specified as dependencies**
    - Application's config/ivy.config file
  - **Dependencies are inherited into your application**
    - Can view inherited artifacts with Virtualized Directory Viewer

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.

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

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.

Zero Modules

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

```
<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 inherit all assets (scripts, static files, java classes) from their dependencies**
- **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 implemented 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…

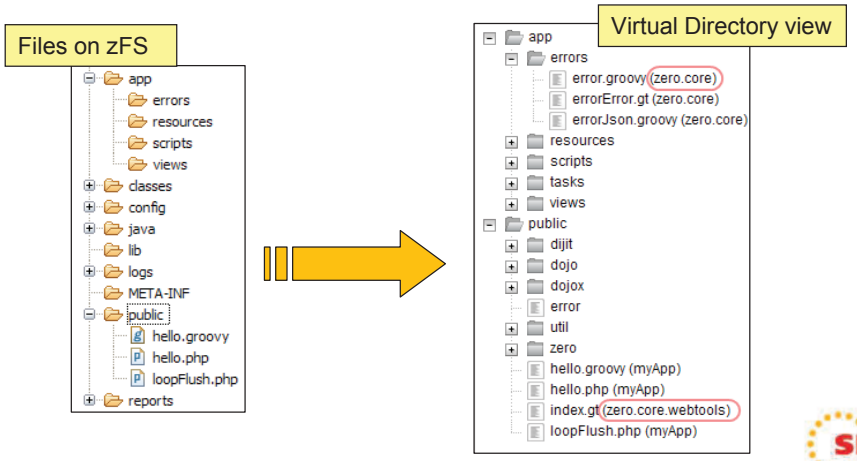
- **An application must be "resolved" before the it can be used. Resolving an app means:**
  - Locating its dependencies & determining exactly which versions to use.
  - Possibly retrieving them from a remote repository, if they are not found in the app's "workspace" or the CLI's local repository.
-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



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 follow-on 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 in a script called "people.groovy" or "people.php" in the /app/resources directory. If this script provides an implementation of the function corresponding 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 (ZRM)

- **Model application data**
  - Application Database focus
  - Robust framework for persistence, validation, and serialization
  - Constrained set of APIs encourages a RESTful application architecture
    - Supplied functions for GET, POST, PUT, DELETE
    - Parses input parameters
    - Obtains a database connection, makes SQL call
    - Processes results
    - Returns results to client

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 the Atom Publishing Protocols.

CICS DS: ZRM

- **Zero Resource Model (ZRM)**
- **“Black Box” implementation, lots assumed/done for you**
- **Place dependency in application’s “`config/ivy.config`”**

```
<dependency name="zero.resource" org="zero" rev="[1.0.0.0, 2.0.0.0["/>
```

- **Uses ‘Derby’ by default, but can use other databases**
- **Incorporate dependency**

```
zero resolve
```


Notes:

- ZRM (Zero Resource Model) is sometimes referred to as a ‘black box’ implementation. It does a lot for you ‘under the covers’, however for many situational applications, the ‘built-it’ way of doing data access is just fine, and allows you to have data access with almost zero effort.
- You only need to add the ‘zero.resource’ dependency, and if no other database is specified, you default to using the embedded version of the Derby database supplied with CICS Dynamic Scripting. You can use other databases with ZRM, and the upcoming slides show how to use DB2 with your CICS Dynamic Scripting application.
- If needed, you can have finer control of your database access (which we will also discuss in future slides), but for many situational applications Derby and ZRM are sufficient.
- Once you have updated the ivy.config file, you will need to invoke a zero resolve command from your application’s home directory.

ZRM Development Life Cycle

app/models/people.json

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

app/resources/people.groovy

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

Invoke from command line

```
/zeroapps/MyApp$  zero model sync
```

app/models/fixtures/initial_data.json

```
[
    {
        "type": "people",
        "fields": {
                "first_name" :   "Alice",
                "last_name" :   "Rogers",
                "location"   :   "Seattle"
        }
    },
    {
        "type": "people",
        "fields": {
                "first_name" :   "Bill",
                "last_name"  :   "Stevens",
                "location"   :   "Seattle"
        }
    },
    {
        "type": "people",
        "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. There are several assumptions if you take this approach, and we will discuss these assumptions in the next few slides.
- On the right is an illustration of how to load initial data into the data table.
- For testing, you may need to reset the values in the table. A 'zero reset' command drops the table, redefines the table, and loads the initial values.

CICS DS: ZRM: Database Definition

- **Columns are added for:**
  - **"id"** `INTEGER PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY (SART WITH 100, INCREMENT BY 1)`
  - **"updated"** `TIMESTAMP NOT NULL`
- **Column Names: cannot contain Java/Groovy/SQL reserved words, hyphens, spaces, must start with alpha character**
- **Column types:**

```
string [max_length | format]  (e.g. large, email, phone,
     region, any)
boolean
date
date-time
decimal [max_digits & decimal_places]
integer
float
```

- **Column options:**

```
required, label, description, default_value, options
```

- **More…. See Project Zero documentation**

Notes:

- For ZRM database definitions, two columns are added in addition to the columns you specify. An 'id' column is added with the characteristics show on the slide plus an 'updated' field that is a timestamp.
- There are some restrictions for the column names as shown on the slide.
- Column types can be string, boolean, date, date-time, time, decimal, integer, and float
- A 'string' can optionally include max_length, The 'format' parameter is used for validation and can be large, email, phone, region, and any.
- A 'decimal' can have max_digits and decimal_places
- The 'date' can have auto_create and auto_update
- Optional field values are required, label (if you don't want the ZRM assigned label), description, and default_value.

CICS DS: ZRM: REST:Database Access

- **Collection**

  `http://host:port/resources/<collection>[/pathinfo][query]`

- **Examples:**

  `…/resources/people`
  `…/resources/people/100`

  `…/resources/people?firstname__equals=Alice`
  `…/resources/people?firstname__contains=ice`
  `…/resources/people?firstname__startswith=A`
  `…/resources/people?start=5&count=5`
  `…/resources/people?firstname__contains=ice&start=5&count=5`
  `…/resources/people?order_by=lastname,firstname`
  `…/resources/people?id__lt=300`

  `…/resources/people?format_as=atom`
  `…/resources/people?format_as=json`

Many Filters, see Project Zero Documentation

Notes:

- This slide shows some of the REST interface options when you expose your table using the "ZRM.delete()" approach.
- This is just a handful of the options, so you will want to look at the Project Zero documentation for a comprehensive look at all the built-in capabilities of the REST interface offered by ZRM.
- There were some slides earlier in the presentation that covered REST.

CICS DS: ZRM: REST:Database Access

- JSON or Atom - can customize rendering (e.g. calculated fields)
- Incoming data (i.e. POST and PUT)
  - JSON (`application/json`)
  - Atom (`application/atom+xml`)
- Can request metadata (table layout in JSON format)
- Can make collections read-only
- There is a Dojo REST table widget – See the employee data sample from the Project Zero web site
- If you don't want a REST interface, you can use ZRM APIs in your application (see next slide)
- More…
  - See Project Zero documentation

Notes:

- There are options to customize data provided by ZRM, details are in the Project Zero documentation.
- When updating database information via the ZRM REST interface, your Web pages can send in data in JSON or Atom format. You just need to set the media type so the knows your data format.
- You can request table meta data.
- You can tell ZRM that your data collection (table data) is read only.
- The ZRM REST interface is great if you want to access data from your Web pages, however sometimes you would like access to ZRM data from within your application. For those situations, there is a simple API available for you to use. Some examples of using the ZRM API are provided on upcoming slides.
- Again we are just scratching the surface of ZRM and there is a lot more information in the Project Zero documentation.

CICS DS: ZRM: API….

```
// retrieve default collection for model 'people'
Type collection = TypeCollection.retrieve('people')

// create new member in collection, returns Member
Member newPerson =
      collection.create(firstname: 'Mickey', lastname: 'Mouse')

// update member in collection, returns Member
newPerson.location = 'Disney World'
newPerson = collection.update(newPerson)

// delete member from collection
collection.delete(newPerson.id)

// retrieve member from collection
aPerson = collection.retrieve(105)

// list all collection results
List<Member> all_people = collection.list()

// get a filtered collection
Collection somePeople =
        collection.filter(firstname__startswith: 'Mi')
```

More APIs available, see Project Zero Documentation

Notes:

- For access to ZRM data from within your application, you can use the ZRM API.
- This slide lists some of the ZRM API.
- As you can see, you can work with the data in your database as a ‘collection’ of data. You can create members of the collection, access members of the collection, update members of the collection and delete members of the collection.
- When you access members of the collection, you can apply filters, specify an order sequence and other options, similar to the capabilities available via the REST interface.
- This is just a some of the ZRM API options, see the Project Zero documentation for a comprehensive list of capabilities.

ZRM line commands

- "`zero model sync`"

  Creates database artifacts and loads data from the application's `initial_data.json` file

- "`zero model reset`"

  Restores database to state just after the model sync (equivalent to dropping database and using zero model sync)

- "`zero model loaddata`"

  Reads data from one or more JSON files

- "`zero model dumpdata`"

  Writes database data in JSON format to one or more files

- "`zero model sql`"

  Outputs generated statements to create and drop database tables (using information in your application's `app/models` folder)

Notes:

- This slide list some of the commands available for working with your ZRM data.
- The zero model sync command was discussed earlier in the presentation and is used to create the database and load initial data.
- The zero model sync does a drop of your ZRM tables, defines the tables, and loads your initial data.
- You can dump your ZRM data in JSON format with the zero model dumpdata command and load the data from JSON formatted files using zero model loaddata.
- You can use the zero model sql command to output the statements used to define your tables.
- If you want some help on the zero model commands, you can zero help model sync (or look at the Project Zero documentation)

CICS DS: Database Support

- **The 'zero.data' module is a thin layer on top of the IBM pureQuery Runtime, on top of JDBC**
- **DB2 UDB for z/OS V8.1**
- **DB2 9.1 for z/OS**
- **Apache Derby v10.3.2.1 and V10.3.1.4**
- **Some databases with type 4 database drivers may work…**
  - **CICS doesn't know you are using a type 4 driver, so be cautious on transactional requirements**

You can use ZRM with any supported database

Notes:

- See the product documentation for a list of supported databases.
- Although you may be able to get databased with type 4 drivers to work with your CICS Dynamic Scripting application, CICS won't know your are using that database, so CICS won't be able to help your with transactionality issues. DB2 uses that CICS-provided DB2/CICS interface so you will have transactionality when working with DB2 in your CICS Dynamic Scripting applications.
- If you do the minimum and only add the zero.resource dpendency to your ivy.config file, you will be using the embedded Derby database, however you can use ZRM with DB2 also, you just need to do some configuration work as described later in the presentation.

CICS DS: Derby Database Support

- **Derby is an 'embedded' or 'networked' database**
- **From Apache Foundation**
- **Database artifacts are persisted out to a specified directory**
- **No CICS resource definitions involved**
- **Add dependency to `config/ivy.xml`**
- **(optional for embedded) specify database characteristics in the `config/zero.config` file**
  - **Location**
  - **Options**

Notes:

- Derby is an open-source database provided by the Apache Foundation.
- Derby comes with CICS Dynamic Scripting and when using ZRM requires no configuration.
- You can, however, provide input to Derby configuration (indicated on the next few slides)

CICS DS: Derby Database Dependencies

- "`config/ivy.xml`" for Derby:

```
<dependency name="zero.data" org="zero" rev="[1.0.0.0, 2.0.0.0["/>
<dependency name="derby" org="org.apache.derby" rev="10.3+"/>
<dependency name="derbyclient" org="org.apache.derby" rev="10.3+"/>
```

- Use the '`zero resolve`' command

- For the above dependency, use the line with 'derby' if using imbedded, use the line with 'derbyclient' if using networked.

Add the "zero.resource" dependency if you want to use ZRM

Notes:

- If you want to use Derby with ZRM, you only need to add the zero.resource dependency.
- If you are just going to use zero.data, then you will need to add the Derby dependency to your ivy.xml file.
- Any time you update your config/ivy.xml file you will need to stop your application, do a zero resolve, then start your application.

CICS DS: Derby Database Driver

- (optional for embedded version) Specify the driver and database location
- In the application "`config/zero.config`" (for the Derby database):

```
/config/db/mydb = {
  "class" : "org.apache.derby.jdbc.EmbeddedDataSource",
  "databaseName" : "db/mydb",
  "connectionAttributes" : "create=true"
}
```

- Create=true - create database if it doesn't already exist
- Verify connection (from `$APP_HOME`)…

```
zero validatedb mydb
```


Notes:

- By default, the embedded Derby database data will be persisted to the db/resource directory in your application. If you want the data persisted to a different directory, you will need to add entries to your config/zero.config directory.
- If you want to test the connection to your database, you can use the zero validatedb command
- See the Project Zero documentation for a list of all Derby database options.

CICS DS: Adding DB2 Database Support

- DB2 datasets in CICS region STEPLIB (SDSNLOAD and SDSNLOD2)
  - Check CICS InfoCenter for prereq PTFs/APARs
- DB2 resource group in CICS startup list
- CICS resource definitions as usual
  - DB2CONN
  - DB2ENTRY
- By default we are running under tranid 'ZPIH', but that can be changed
- Can run under CICS region userid, userid in config file, userid established with SSL client certificate exchange or HTTP Basic Authentication

Notes:

- This and the next few slides talk about using DB2 with CICS Dynamic Scripting applications.
- You will need to do the normal CICS setup when using DB2. This includes adding PDSs to your CICS region STEPLIB, adding the group containing DB2 program definitions and other resource defintitions in your CICS regions startup list.
- Additionally, you will need to make the normal DB2CONN and DB2 ENTRY definitions.
- By default your application's requests will run under the ZPIH transaction
- We're not going go any further into CICS/DB2 configuration, but it should be business as usual.

CICS DS: DB2 Driver Access

- Each CICS Dynamic Scripting application: adding environment variables in the JVMProfile file is not enough
  - DB2 Jars to `$APP_HOME/lib` directory
    - db2jcc.jar, db2jcc_javax.jar, db2jcc_license_cisuz.jar
    - Can use symbolic links
  - DB2 JNI libraries in `$APP_HOME/lib/s390/zos`
    - libdb2jcct2zos.so
    - Can use symbolic link
- Check InfoCenter for prereq PTFs/APARs

Notes:

- Each CICS Dynamic Scripting application will need to point to the DB2 drivers, license file, and the JNI files needed by the driver.
- Make symbolic links as specified on the slide for each application
- As always, check the CICS InfoCenter for needed prereq PTFs/APARs

CICS DS: Adding Database Dependencies

- "`config/ivy.xml`" for **DB2**:

```
<dependency org="zero" name="zero.data" rev="[1.0.0.0, 2.0.0.0["/>
<dependency org="zero" name="zero.cics.db2" rev="[1.0.0.0, 2.0.0.0["/>
```

- **Use the '`zero resolve`' command**

(note that the zero resolve also processes the files (links) you placed in the lib directory on the previous slide – i.e. do the zero resolve after you have added the dependencies and symbolic links)

Add the "zero.resource" dependency if you want to use ZRM

Notes:

- To add the DB2 dependency to your CICS Dynamic Scripting application, add the zero.cics.db2 dependency to your config/zero.config file, then do a zero resolve.

CICS DS: DB2 Database Driver

- "zero.data" is a layer on top of Java's standard JDBC interfaces
- In the application's "`config/zero.config`" (for DB2):

```
/config/db/mydb = {
    "class" : "com.ibm.db2.jcc.DB2DataSource"
}
/config/resource/dbKey = "mydb"
```

- Can specify schema in the above config (after driver)

```
"currentSchema" : "WSPOT03"
```

- Verify connection (from `$APP_HOME`)…

```
zero validatedb mydb
```


Notes:

- This slide shows the elements to be added to your application's config/zero.config file when using DB2.
- Consult the Project Zero, CICS, and DB2 documentation for various options
- You will want to test your connection to the DB2 database. You can use the zero validatedb command.

CICS DS: Number of DB2 Connections

- **Previously: there was a connection limitation of 1 DB2 connection per JVMServer**
  - **Could serialize access or specify one thread per JVMServer to accommodate this 'situation'**
- **Now: one connection for each task in a JVMServer**
  - **PTF UK69637, UK696555, and UK69654 for 1 DB2 connection per task instead of 1 connection per JVMServer**

Notes:

- There was previously a limitation of one DB2 connection per JVMServer, but with the appropriate PTFs you can now have 1 connection per task versus the previous 1 connection per JVMServer.

CICS DS: DB2 or Derby: Database Use

- Could use ZRM REST, or ZRM API (previous slides)
- Can get/use a connection:

```
// Groovy code:
import zero.data.groovy.Manager
…
def data = zero.data.Manager.create('mydb')
def results = data.queryList('SELECT * FROM table')
```

- "`mydb`" is the '`dbKey`' and corresponds to the example on the previous slides
- "`zero.data`" will match the '`dbKey`' and use the configuration properties from the Global Context (i.e. the items you placed in your `zero.config`)
- After you have a '`Manager`', you can use various APIs including raw JDBC, however '`zero.data`' APIs are more friendly

Notes:

- When using the Derby or DB2 database, you can use ZRM and/or the ZRM APIs
- Additionally, you can use zero.data APIs to access your database data

CICS DS: Options for Database Creation

- **Use existing table, have your DB2 admin create the table for you, use your regular backup procedures, etc**
- **Use ZRM (Zero Resource Manager)**
  - **Discussed previously**
- **Code your own table create**
  - **Use '`zero runsql [dbKey] file`'**

```
CREATE TABLE employees (
  username varchar(32) NOT NULL,
  firstname varchar(16) NOT NULL,
  lastname varchar(16) NOT NULL,
  location varchar(64) NOT NULL,
  phonenumber varchar(16) NOT NULL,
  PRIMARY KEY  (username)
);
```


Notes:

- Earlier in the presentation we talked about defining database tables using ZRM and the zero model sync command
- Most customers will have a DB2 administrator and fairly ridge procedure for getting access to DB2 tables, even in a test environment. If you are in this category, no problem, you can have your CICS Dynamic Scripting application use an existing DB2 table(s).
- As an additional option, you can specify SQL commands to define tables, alter tables to add new columns and other SQL commands in files. You can then execute the commands using the zero runsql command.

CICS DS: SQL

- **In code below, use queryFirst() if you want the first or only row, use queryList() if you want nn rows**

```
// get input parameter
string username = request.params.employeesId[]

// get DataManager for specified database
def data = zero.data.groovy.Manager.create('mydb')

// Retrieve employee record via Data Zero
def employeeRecord = data.queryFirst
 ("SELECT * FROM employees WHERE username=$username")
```


Notes:

- This slide is an example of using the zero.data API. You will notice that it allows you to leverage your SQL skills.
- The first line of code just gets an input parameter from a request that came in over the Web. An explanation of what's going on here is beyond the scope of this presentation.
- The 'mydb' is the dbkey specified in the config/zero.config file
- There are few different method you will use to make SQL request. A couple of them are queryFirst() if you expect to get one-at-most results returned, or queryList if you want allow for the possibility of having multiple results returned from your SQL

CICS DS: SQL: Prepared Statements

- Can avoid assembling SQL statements from fragments (error prone)
- Can help avoid SQL injection attack

```
def id = request.params.id[]
def result = mgr.queryFirst
 ("select * from employees where id = ?", id)

def id = request.params.id[]
def result = mgr.queryFirst
 ("select * from employees where id = ${id}")

def args = ['tag': '%'+request.params.tag[]+'%']
def result = mgr.queryList
 ("select * from employees where tags like :tag", args)
```


Notes:

- This slide shows three additional approaches when interacting with your database.
- Having a 'prepared statement' is less prone to error than dynamically constructing your SQL from segments and will likely cut down on the possibility of an SQL injection attack where the attacker places SQL in one of your Web browser form fields, which you then place inside your dynamically created SQL statements.

CICS DS: Externalized SQL Statements

- **Can place SQL statements in your "`config/zero.config`"**

```
/config/db/mydb/statements = {
 "GET_ALL" : "select * from employees",
 "GET_BY_TAG":"select * from employees where tags in :tag",
 "ADD":"insert into employees(firstname, lastname)
values (?,?)"
}
```

- **In your application code**

```
// usage example in groovy
def args = ['tag': '%'+request.parms.tag[]+'%']
def result = mgr.queryList('GET_BY_TAG', args)
```


Notes:

- You can, if you want, place all of your SQL statements in your config/zero.config file.
- This has the benefit of having all of your SQL statements in one location will aid in code reviews or inspections/suggestions from your database administrator.
- The code segment on the bottom of the slide shows how you might use the SQL statements in your application. The sample is coded in Groovy.

CICS DS: REST: onList() (do it yourself)

```
def onList() {
  try {
    // Get configured DataManager for data access
    def data = zero.data.groovy.Manager.create('mydb')
    // Retrieve employee records via Data Zero
    def result = data.queryArray('SELECT * FROM employees')
    request.view = 'JSON'
    request.json.output = result
    render()
  } catch (Exception e) {
    if (e.getCause() instanceof java.sql.SQLException) {
      request.status = HttpURLConnection.HTTP_INTERNAL_ERROR
      request.error.message = "The db may not have "+
                              "been initialized."
      request.view = "error"
      render()
    }
  }
}
```


Notes:

- This sample shows how you might implement your own REST interface for a GET request against your collection.
- This code segment would be in a script in your app/resources directory.
- CICS Dynamic Scripting will invoke 'well known' functions in your program depending on whether there was a GET request against the collection, a GET request of a specific member of the collection, or a POST, PUT, DELETE request.
- Notice the use of a 'renderer' to format the data in JSON format.
- You can customize supplied renderers if you want.

CICS DS: REST: onCreate() (do it yourself)

```
def onCreate() {
  // Convert entity to JSON object
  def emp = zero.json.Json.decode(request.input[])
  // Get DataManager for data database
  def data =
        zero.data.groovy.Manager.create('employee_db')
  // Insert employee record via Data Zero APIs
  data.update("""
   INSERT INTO employees (username, firstname, lastname,
   location, phonenumber)  VALUES ($emp.username,
   $emp.firstname, $emp.lastname, $emp.location,
   $emp.phonenumber) """)
  // Set a Location header with URI to the new record
  locationUri = getRequestedUri(false) + '/' +
      emp['username']
  request.headers.out.Location = locationUri
  request.status = 201;       // created
  request.view = 'JSON'
  request.json.output = emp
  render()
}
```


Notes:

- Similar to the previous slide for a GET request against a collection, the is a POST request for a new member of a collection.
- Notice the altering of the URI
- Again, notice the use of a renderer to format the data in JSON format.
- You can customize the supplied renderers if you would like. See the Project Zero documentation.

CICS DS: DB Output in XML Sample

- **In your application's code:**

```
def writer = new StringWriter()
def xml = new groovy.xml.MarkupBuilder(writer)
def data = zero.data.groovy.Manager.create('mydb')
xml.people(xmlns: 'http://xmldata.myco.com') {
  data.eachRow('SELECT * FROM people') {
    row ->
      person {
        firstName(row['firstname']) { }
        lastName(row['lastname']) { }
        location(row['location']) { }
      }
  }
}
println writer.toString()
```

- **Evaluates to:**

```
<people xmlns='http://xmldata.myco.com'>
  <person>
    <firstName>Jerry</firstName>
    <lastName>Cuomo</lastName>
    <location>North</location>
  </person>
</people>
```


Notes:

- The are various data formatting capabilities in CICS Dynamic Scripting, to include XML. This slide shows an example of the Groovy MarkupBuilder.
- Note that the MarkupBuilder also allows you to add attributes to XML tags

CICS DS: Atom Feeds

```
// code to show it can be done…  declare the feed
def atom_feed = [:]
atom_feed.title = 'Sample Atom Feed'
atom_feed.updated = new Date()
atom_feed.entries = []

// loop thru creating members and adding them to the feed
    //declare a member entry in the feed
    def member_entry = [:]
    member_entry.id = '1'
    member_entry.title = 'Title Information'
    member_entry.updated = new Date()
    member_entry.summary= 'Summary Info‘
    // add detail member content from database to next line
    member_entry.content = 'This is detailed content'
    // add the member to the feed
    atom_feed.entries += member_entry


// format and send the feed
request.view='atom'
request.atom.output = atom_feed
render()
```


Notes:

- This slide doesn't show any database interaction, but it does show that there is an Atom renderer, and that it is fairly easy to place your data in a formation that can be understood by the Atom renderer.
- If you are familiar with Atom feeds, the sample will look pretty simple. If you are not familiar with Atom feeds, well, see the next slide as to what is produced from this code.

CICS DS: Output from previous slide

```
<?xml version="1.0"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <id>http://host:8888/resources/aFeed</id>
  <title type="text">Sample Atom Feed</title>
  <link href="http://host:8888/resources/aFeed" rel="self">
  </link>
  <entry>
    <id>http://host:8888/resources/aFeed/1</id>
    <title type="text">Title Information</title>
    <link href="http://host:8888/resources/aFeed/1" rel="self">
    </link>
    <summary type="text">Summary Info</summary>
    <updated>2011-08-04T02:25:37.887Z</updated>
    <content type="text">This is detailed content</content>
  </entry>
  <updated>2011-08-04T02:25:37.887Z</updated>
</feed>
```


Notes:

- This is the ATOM data produced from the previous slide.
- For an Atom feed, there is a feed tag for the ‘root’ element of the document. There are some values about the feed itself, and then there is an entry tag for each member entry of the collection represented by the feed.
- If this was the result of a request of information pertaining to a specific member entry, then ‘root’ element of the XML document would be ‘entry’.
- If you are returning information from CICS in the form of an Atom feed, you are

Where else do I get Data?

- Can use JCICS API (there is a bridge to Java)
  - LINK to a CICS program
  - READ VSAM file
  - READ TSQ
  - etc
- Data to/from CICS resource (field-level access to your PHP, Groovy, or Java program – i.e. getters/setters)
  - JZOS – can analyze 'ADATA' compiler information and generate Java data class
  - RAD's J2C wizards – CICS Java Data Bindings
- Output: Can use CICS DS Renderers (or println/echo)

Notes:

- Since you are running in a CICS environment you are likely to want to LINK to an existing program to get data, or directly access a CICS resource.
- One of the challenges you will have with interacting with CICS resources or LINKing is to bridge between the object-oriented world of Java, Groovy, or PHP, and the series-of-bytes, field-oriented world of CICS resources, COMMAREAs, and channels and containers
- Once you have obtained data from a CICS resource, you can use the CICS Dynamic Scriptin renderers to format the data.

Interfacing with CICS Programs

```
<?php
// Instansiate a COMMAREA representation
// The CustProgCommarea class was created from a COBOL
// data layout using RAD, but could have used JZOS also
$commArea = new Java('com.ibm.ddw.customer.CustProgCommarea');
// Set some data in the commarea by calling method on the class
$commArea->setRequest__type('R');
$commArea->setCustomerId('00000001');
// Use the JCICS class to call a CICS program
$program = new Java('com.ibm.cics.server.Program');
$program->setName('CUSTPROG');
try {
    $program->link($commArea->getBytes());
} catch (CICSException $e) {
    echo $e->getMessage();
    exit;
}
echo "Return value is " . $commArea->getCustomerFirstName();
?>
```


Notes:

- For the code example on this slide, we used the J2C wizards to create a CICS Java data Binding.
- We also could have used JZOS. We would have compiled the target CICS program (CUSTPROG in this case) with the ADATA compiler option. We would have used the ADATA information representing the COMMAREA of the CUSTPROG program as input to the JZOS classes to generate a Java object that represents the COMMAREA (which we would have called CustProgCommarea (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 CUSTPROG program’s COMMAREA.. We then invoke methods on the class to set values (the example invokes the setCustomerID() 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 “CUSTPROG” (because CUSTPROG 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 CUSTPROG 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

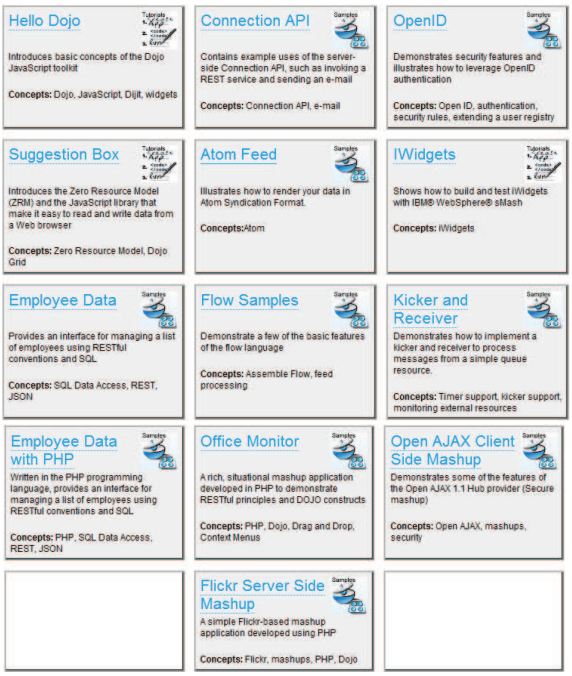
Interfacing with CICS VSAM File

```
<?php
// Used RAD for the CustProgFileLayout class, could have used JZOS
$recordLayout = new
   Java('com.ibm.ddw.customer.datalayouts.CustProgFileLayout');
// the record key for the KSDS VSAM CUSTDATA file
$theKey = '00000001';
try {
  // Use the JCICS class to read from a KSDS VSAM file
  $custFile = new Java('com.ibm.cics.server.KSDS');
  $custFile->setName('CUSTDATA');
  $recordHolder = new Java('com.ibm.cics.server.RecordHolder');
  $readKey = mb_convert_encoding($theKey, "1047", "iso-8859-1");
  $custFile->read($readKey, $recordHolder);
  $recordLayout->setBytes($recordHolder->value);
} catch (CICSException $e) {
  echo $e->getMessage();
  exit;
}
echo "Return value is ".$recordLayout->getCustomerFirstName();
?>
```


Notes:

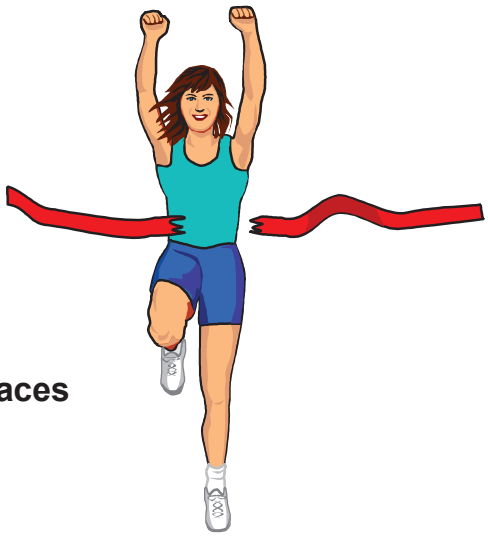
- Like the LINK example, we have created a CICS Java Data Binding that represents the layout of our VSAM file.
- In this example we again use the Java bridge to allow us to use the JCICS classes.
- We are reading a KSDS file, so we instanciate a KSDS object and set it to the name of the VSAM file with which we will interact.
- We create a ‘record holder’ and pass it to CICS on the read method along with the record key.

Tutorials, Samples, and Demos



Summary -- Session #09608

- What is CICS Dynamic Scripting
- Review data-related concepts
- ZRM (Zero Resource Model)
  - Define/dump/load database tables
  - REST access and ZRM API access
- More database options
  - Configuration
  - Defining tables
  - Accessing table data
  - Implementing your own REST interfaces
- Formatting data (JSON, XML, ATOM)
- Accessing Data via the JCICS API



References

- **See my presentation from 2011 Winter SHARE for a CICS Dynamic Scripting Overview (there is a notes page for each slide)**
- **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

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.
- If you don't yet have CICS Dynamic Scripting installed, try installing WebSphere sMash DE (Development Edition), which is free for download and development.