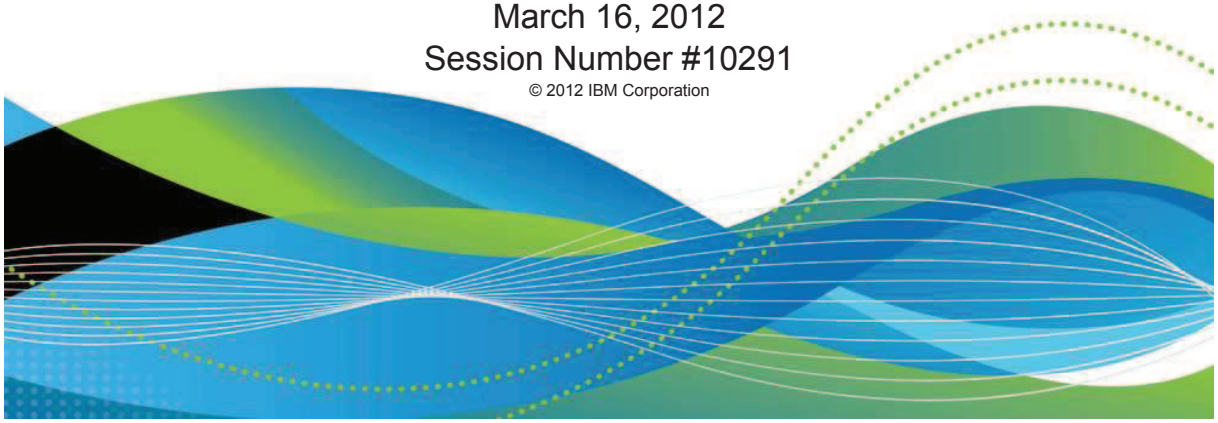
#SHAREorg

CICS Dynamic Scripting: the Presentation Layer

Dennis Weiand
IBM Advanced Technical Skills

March 16, 2012
Session Number #10291



Abstract

- CICS Dynamic Scripting provides an agile environment for quick development of situational applications. Your application's presentation layer (web browser interaction) is an important part of a situational application and you need to know the basics concepts and capabilities to get started.

  This topic discusses the available options when developing the presentation layer of your CICS Dynamic Scripting application, along with the accompanying concepts. Areas of discussion will include serving HTML, Cascading StyleSheets, and JavaScript, along with AJAX, Dojo, RESTful interactions, and security.

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 Oracle, 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

- Web-related Terminology
  - HTTP, HTML, CSS, DOM, JavaScript
  - AJAX, Widgets, DOJO
  - Mobile
- What is CICS Dynamic Scripting (CICS DS)
- Web application design
  - The typical design I use
  - Maintaining state
- Getting CICS DS Request information
- CICS DS Response Rendering
  - Static web pages, CSS, JavaScript
  - Dynamic Web pages
    - PHP Rendering
    - Groovy Rendering
    - REST/AJAX
  - Error Responses
  - Implementing REST interfaces
- Security
- **NOTE:** For an overview of CICS DS or for a look into storing and accessing data, see my previous SHARE presentations

Credits

- Get information from one source and that is called plagiarism. Get information from more that one source and that is called research.
- Lots of great information on Project Zero, WebSphere sMash, and CICS Dynamic Scripting…. I start with….
  - "Getting Started with IBM WebSphere sMash" – by Ron Lynn, Karl Bishop, Brett King
  - WebSphere sMash and CICS TS V4.2 InfoCenter
  - SG24-7924 - Introduction to CICS Dynamic Scripting
  - www.w3schools.com (tutorials on most anything "web")
  - Wikipedia (how did we live without Wikipedia?)
  - Roy Fielding year 2000 Doctoral thesis where he describes REST (yes I read the whole thing)
  - The "Internet" (like all the great people that ask and answer questions on forums)
- **WARNING:** This presentation contains no original thought. It is a summary of information from the many sources of information on this topic (although I must state that I included a couple of my opinions).

Web-related Terminology

- **HTTP – HyperText Transfer Protocol**
- **URL – Uniform Resource Locator**
- **HTML – Hypertext Markup Language**
- **CSS – Cascading Style Sheets**
- **DOM – Document Object Model**
- **JavaScript – Scripting Language that runs in the web browser**
- **Dojo – Open source JavaScript routines**
- **AJAX – Asynchronous JavaScript And XML**
- **REST – light-weight data transfer that leverages the HTTP protocol**
- **Mashup – data from multiple sources on one web page**
- **Widget – Web gadget – small application**
- **MIME Types – Multi-purpose Internet Mail Extension**

(will elaborate on some of these on next few slides)

Notes:

- This presentation is intended to address the presentation layer aspects of CICS Dynamic Scripting. We are primarily discussing concepts.. i.e. enough information so you can effectively use the *real* sources of information on CICS Dynamic Scripting and various aspects of providing a presentation layer for a web-based, HTTP-based application.
- The CICS Dynamic Scripting Feature Pack V1.0 is a no-charge feature of CICS TS V4.1 and the CICS Dynamic Scripting Feature Pack V1.1 is a no-charge feature of CICS TS V4.2.
- Before we start into CICS Dynamic Scripting, we need to get some terminology under our belt. This slide covers some of the main acronyms I will refer to in this presentation
- I will provide a high level definition of these terms on this slide, but since understanding these terms is crucial to understanding the rest of the presentation, I will also have an upcoming slide dedicated to some of these terms.

URL – Directing requests using HTTP

- Scheme – http or https
- Host name or address
- Port (optional)
- Path
- queryString (may or may not see these on URL)
  - Name=value pairs separated by an '&'

```
http://demomvs.demopkg.ibm.com:8091/account?name=Dennis&state=TX

scheme://host:port/path?queryString
```


Notes:

- HTTP requests are routed using a URL. A URL (Uniform Resource Locator) specifies the location of the server, and optionally, some information that should be passed to the server.
- The URL consists of a scheme, which can be http or https. An 'http' specification indicates that the flows are not to be encrypted, but an 'https' specification indicates the flows should be encrypted.
- Next comes the host which is a DNS (Domain Naming Service) name, or a dotted-decimal address. A DNS name is just a 'friendly' name for a dotted-decimal address. When a DNS name is specified, a DNS server will be contacted to resolve the specified name to dotted-decimal address.
- The (optional) port is separated from the host name by a colon. If no port is specified and the scheme is http, the port used will be 80. If no port is specified and the scheme is https, the port used will be 443.
- Following the optional port is the path. The path can be from 1 to 255 characters, and its function is to tell the server what to do with the request.
- Following the path is an optional query string. If a query string is present, it is separated from the path by a '?' character. The query string is a set of name/value pairs (separated by an '&') that are to be sent to the server.

HTTP – the Internet protocol

- A Request…

  - Request Line: method absolute_path http_version
    ```
    post /account/lookup HTTP/1.1
    ```
  - Headers: name:value;name:value;etc:etc
    ```
    Accept: */*;Accept: image/jpeg;Content-length: 44
    ```
  - Blank Line: Carriage return Line feed twice
  - Body: URL encoded forms data: name=value pairs
    ```
    field1=stringa&field2=stringb&field3=stringc
    ```

- A Response…

  - Status Line: HTTP_version status_code response_phrase
    ```
    HTTP/1.1 200 ok
    ```
  - Headers: name:value;Name:value;etc:etc
    ```
    Content-type: text/html;Content-length: 54
    ```
  - Blank Line: Carriage return Line feed twice
  - Body: HTML tags and text
    ```
    <html><title><A Sample</title><h1>Sample 1</h1>...
    ```


Notes:

- Web browser requests and responses flow over HTTP (Hypertext Transfer Protocol). If you were to look at an HTTP flow 'on the wire', you would see a request line indicating the type of request followed by a URI, and an indication of the HTTP level in use.
- The type of request is called the 'method', which on the previous page is 'POST'. For a request from a web browser for HTML pages, you normally see a 'POST' or a 'GET'. For Web service you almost always see a 'POST'. For REST request, you will see 'GET', 'POST', 'PUT', or 'DELTE'. We will touch on REST requests later in the presentation.
- The path indicates the target of the request, with the method indicating the action on the target.
- Following the request line are headers. Headers are additional data that further qualifies the request. The Header may indicate the capabilities of the eeb browser (like what files it can accept), plus other characteristics (like the content length or the language setting of the web browser). Headers can also contain security information in the case of HTTP Basic Authentication, or information to be stored in the web browser in the case of Cookie data.
- Optionally, following the headers is the HTTP body. When the web browser sends information to the server with a 'POST' request, the body is a set of name/value pairs. When the server sends information to the web browser, the body normally contains HTML.
- Initially, it may seem a bit confusing with HTTP (the underlying protocol) has a 'body'; HTML has is transferred over HTTP also has a body; and for a web service, in the SOAP envelope you will find a body. As you get into the various types of HTTP flows, you will understand their simplicity with most everything having a body that contains the user data being transferred. HTTP is general transport, so the content of the body will vary depending on the type of client and the type of request. In the preceding slide the body is said to have name value pairs. This is true for a 'POST' method from a web browser. For a web service, the body will contain the SOAP envelope, and for a REST request, the body will normally contain XML or JSON (JavaScript Object Notation). More on this later.
- A response to the web browser from CICS (a web server in this case) will normally consist of a status line indicating the response to the request (200 is a normal response), headers indicating the type of data being sent back from the server (the MIME type) and other information, and a body that contains data related to the response.
- MIME stands for Multipurpose Internet Mail Extension and is used to signify the type of file, such as XML, JSON, GIF, etc.
- There are different types of header values that can be returned such as the length, Cookie data, and other control information. See the HTTP specification for a list of all headers.
- The body of the HTTP transmission sent to web browser to display a Web page is normally some HTML that provides a response to the request.
- The HTML will reference any other files needed to properly display the HTML (like a Cascading Style Sheet, or graphics).

HTML – Hypertext Markup Language

```
<html>

<head>
<title>Sample HTML Page</title>
<script language="JavaScript" src="theme/menu.js"></script>
<link rel="stylesheet" type="text/css" href="css/screen.css" />
</head>

<body>
<form method="post" action="/account/lookup">
<p>Dear Dennis,
<p>Please enter your address below and press the enter button:
<p><textarea name="address" rows="8" cols="40"></textarea>
<p><input name="submit" type="submit" value="Enter">
</form>
</body>

</html>
```


Notes:

- HTML (HyperText Markup Language) in a markup language interpreted by web browsers. The general format is: `<tag atribute1="value1" attribute2="value2">content</tag>`
- A physicist named Tim Berners-Lee, a contractor at CERN (European Organization for Nuclear Research) prototyped a system for researchers to use and share documents. Berners-Lee, in 1989, proposed a Internet-based hypertext system, and in 1990 wrote a prototype of the system that used HTML. The first publicly available HTML description was available in 1991.
- HTML elements define areas of a web browser page, header levels, tables, form elements for data entry, and much more
- HTML elements can be specified with style characteristics such as color, but instead of hard coding the characteristics in-line, most often, the characteristics are often specified in a Cascading Style Sheet (CSS). Cascading Style Sheet information can be inserted in-line in an HTML page, or the HMTL page can reference a file containing the CSS information, which causes the web browser to request/load the CSS file:

  ```
  <link rel="stylesheet" href="http://example.com/css/style.css" type="text/css" />
  ```

- This HTML page also references a JavaScript file that the web browser is to load. JavaScript is a scripting language that can run in the browser. The running JavaScript functions is triggered by events that take place at the web browser such as a button press, mouseover, key down or key up, and many more.

Cascading StyleSheets

- **Used to indicate web page formatting**
- **One place to specify style information vs. 'hard coding' the information in HTML**
- **CSS can be in-line or in a separate file**
- **To reference a separate file you can add:**

```
<link rel="stylesheet" href="http://example.com/css/style.css"
type="text/css" />
```

- **Example:**

```
h1 { color: white;  /* tag selector */
     background-color: orange !important;
}
#para1 { text-align:center; color:red; }   /* id selector */
.center {text-align:center; }              /* class selector */
p.center {text-align:center; }     /* tag-specific class selector */
```


Notes:

- CSS (Cascading Style Sheet) are used to specify the look and formatting of web pages by supplying a set rules pertaining to the characteristics of HTML tags. They are typically used to specify elements such as layout, color, and font, and can specify many different tag characteristics.
- CSS started to become supported by web browsers in the late 1990s.
- A style sheet consists of a list of rules, consisting of one or more selectors and a declaration block.
- General syntax is (see slide for examples):

```
selector [, selector2, ...] [:pseudo-class] {
 property: value;
 [property2: value2;
 ...]
}
/* comment */
```

- Cascading Style Sheet information can be specified in-line in an HTML page with a <style> tag, or you can refer to a file containing Cascading Style Sheet information, which causes the web browser to request/load the CSS file:

```
<link rel="stylesheet" href="http://example.com/css/style.css" type="text/css" />
```

- You often use more than one CSS. This was anticipated and hence the use of the word 'cascading' in the name. If the same selector is used in multiple referenced CSS files, the last one in wins.

JavaScript

- **A scripting language that runs in a web browser**
- **Syntax was influenced by Java (but is not the same)**
- **Commonly used to validate user input, dynamically add, delete, change, show, hide (etc) elements on a web page**
- **Can be in-line or in a separate file**
- **Considered a 'common' technology in Web 2.0**
- **Its use is increasing**
- **Used for AJAX (Asynchronous JavaScript And XML)**
  - Make synch/asynch request to server for information
  - Response information is added to the web page

Notes:

- JavaScript is a programming language that runs in a web browser that can be used to programmatically affect the web page content. It is typically used for validating user input and can dynamically change cursor placement, add remove web browser elements, plus much more.
- JavaScript 'functions' can be invoked from 'events' at the web browser such as moving your mouse cursor over a web page element, clicking a button (and much more).
- JavaScript, originally called Mocha, was developed by Brendan Eich of Netscape. Its name changed from Mocha, to LiveScript, to JavaScript (because of its strong influence by the Java programming language). JavaScript was released as part of the Netscape Navigator web browser in 1995.
- Using JavaScript, you can cause the web browser to make an asynchronous request to a server for information. JavaScript can then display the returned information, or take appropriate actions. This technique of making asynchronous requests is very common in Web 2.0, and is referred to as AJAX (Asynchronous JavaScript And XML)
- In today's environment, it is rare to receive a web page that doesn't use JavaScript, with some web pages making very extensive use of JavaScript.

JavaScript Example

HTML form validation using JavaScript

```
function noEntry() {
        if (document.theForm.firstName.value.length<1) {
                alert("The client's first name may not be empty.");
                document.theForm.firstName.focus();
                return false;
        } else if (document.theForm.state.value == "--") {
                alert("Please use the state pull-down menu to "+
                        "specify the Client's state of residence.");
                document.theForm.state.focus();
                return false;
        } else return true;
}
```

The above JavaScript could be triggered by clicking the 'submit' button associated with the form

```
<form name="theForm"  method="post"
action="/inv/scripts/INVSite.php" onsubmit=" return noEntry(); ">
```


Notes:

- This slide shows an example of a JavaScript function that would be used for form validation.
- The JavaScript function would be invoked when the 'submit' button associated with this form is clicked.
- The form will be submitted if the JavaScript routine returns true
- "document" refers to the web page, "theForm" is the name associated with the form, "firstName" is a field on the form, and "length" is an attribute of the field that is the size of the data entered into that form field. So document.theForm.firstName.length is the size of the data typed into the firstName field of theForm on this web page.
- "alert" causes a pop-up box with the indicated text

AJAX Example

Sample of JavaScript that causes an AJAX request:

```
function emptyQuoteFile() {
    xmlhttp=new XMLHttpRequest();
    xmlhttp.onreadystatechange=function() {
        if (xmlhttp.readyState==4) {
            if (xmlhttp.status==200) {
                document.getElementById("vehicleQuoteResponse").
                   innerHTML="<b>"+xmlhttp.responseText+"</b>";
            } else {
                document.getElementById("vehicleQuoteResponse").
                   innerHTML="<b>ERROR: Status: "+xmlhttp.status+
                     "Contact support if this continues</b>";
            }
        }
    }
    xmlhttp.open("GET","/insurance/quote/rest?DoFileClear=yes",false);
    xmlhttp.send();
}
```


Notes:

- This slide shows an example of a JavaScript function that is an AJAX request
- To make an AJAX request, you instanciate an XMLHttpRequest object
- Towards the bottom of the code you will notice a JavaScript method invocation for a method named 'open()'. You specify a URL (with optional query string), and the HTTP method (GET in this case). At the end of the open() you see a boolean value of 'false'. This indicates that the request is to be synchronous and not asynchronous.
- The send() method specifies that the request should be sent
- Back towards the top you see an xmlhttp.onreadystatechange=function() { …….. This associates an 'anonymous function' with the onreadystatechange() method of this XMLHttpRequest object.
- The XMLHttpRequest object goes through various state changes as it makes the HTTP request, waits for a response, receives the response, or times out. The onreadystatechange() method (in this case this anonymous function) is invoke everytime this XMLHttpRequest object goes through a state change.
- The anonymous function tests to see of the readystate is a 4 which indicate the this XMLHttpRequest object has received a response. If the readystate is a 4, this XMLHttpRequest object test if the returned HTTP status code is 200, and if true, places the response text into a text field on the web page named vehicleQuoteResponse.

Widgets

- **Short for “Web Gadget”**
- **Small application that runs in an area in the web browser**
- **Can often take input, provide output to page or other widgets**
- **Example: mortgage calculator, calendar, etc**

```
<script type="text/javascript">
function createAtomFeedTable() {
  var atomFeed=new ATOMFeedObject();
  atomFeed.atomFeedLocationDiv=document.
       getElementById("atomFeedDiv");
  atomFeed.setFeedURL("http://host:port/atom/custdata/feed");
}
</script>
<body onLoad="createAtomFeedTable();">
<center><h1>Generic CICS ATOM Information Display</h1></center>
<div id="atomFeedDiv" align="center"></div>
```


Notes:

- “widget” is short for web gadget, and is usually just a JavaScript function. This JavaScript function normally represents a small application that runs in an area of the web browser.
- An example of a widget might be a mortgage calculator.
- A good widget allows for dynamic input, and can provide output to the web page or to other widgets.
- In this code snippet, you see an onLoad= attribute on the body tag. This indicates that the specified function should be invoked after all parts of the web page is received and just before the web browser is to render (figure out how to display) the page.
- The createAtomFeedTable() function instaciates the ‘widget’, which is called ATOMFeedObject, and sets the location of the output of the widget (which is a table of information), and the location (URL) of the Atom Feed.
- Note in the last line of the code snippet, we have defined an area on the web page with the tag with a name of atomFeedDiv. We have specified the name of this div to the ATOMFeedObject widget. The ATOMFeedObject will create a table of the data passed to it, and place that table in the “atomFeedDiv” div.
- The setFeedURL() method of the ATOMFeedObject widget is written to cause the widget to make an AJAX request to the specified URL, parse the response data, and put the response data in a table at the specified location (the “atomFeedDiv” div).

DOJO

- Open source modular JavaScript library
- Intended to provide common functions (e.g. calendars, sortable tables, dynamic charts, etc)
- Cross-platform (renders similarly on all web browsers)
- Provides mobile look and feel of a 'native' app to HTML-based pages
- My Opinion: Use Dojo
- Started in 2004 (Alex Russel, Dylan Schiemann, David Schontzler, and others)
- IBM and SUN joined the Dojo foundation in 2005

Notes:

- Dojo is an open-source JavaScript library that provides a higher-level interface to common elements or functions used with a web brower.
- Dojo includes calendars, tables, and much, much more.
- An exciting part of Dojo is that it accommodates the differences between web browsers and web browser levels. This means that if you define an element on a web page with Dojo, it will look the same on every web browser.
- Dojo's mobile support allows you to define web pages that when displayed on a smartphone or tablet (e.g Android phone, iPhone, iPad, etc) have a look and feel that is close to a native app.

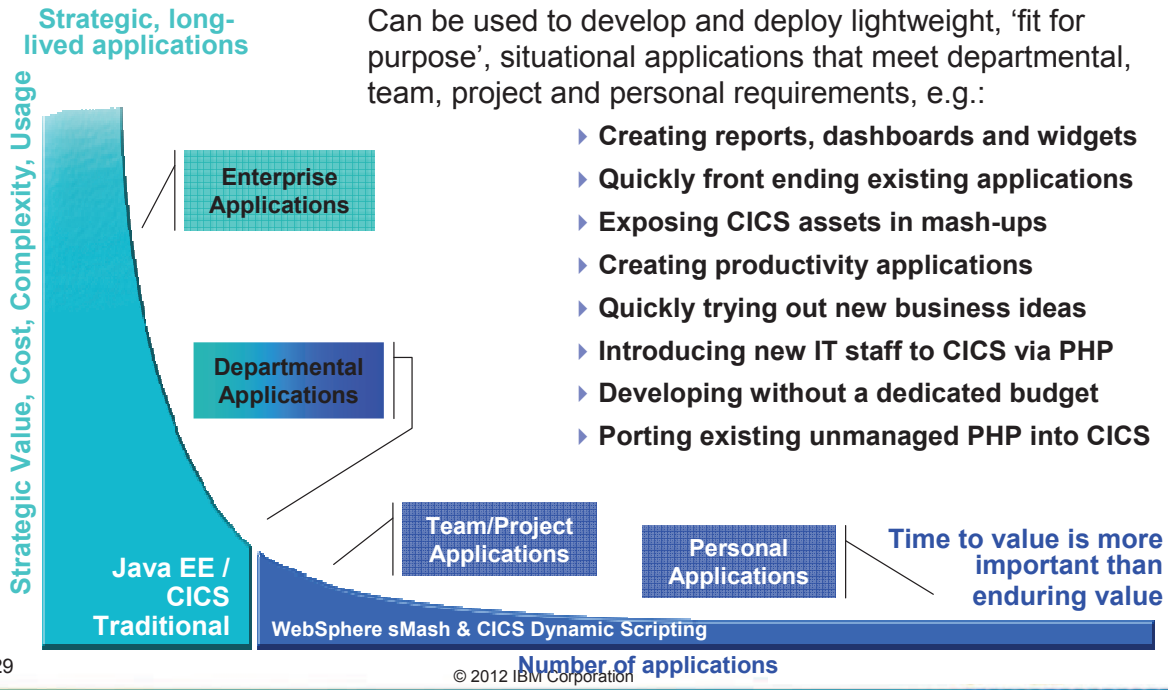
Debugging Web Pages

- Use a good editors (HTML, CSS, JS, etc - avoid errors)
  - e.g. RDz, RAD, Eclipse
- Watch out for browser cache
  - Can turn off browser caching
  - Usually holding 'shift' and clicking the refresh button is good enough, but still beware (I can't tell you how many hours I have lost on this)
- Debug consoles and debuggers – part of most web browsers
  - Firefox (Mozilla), Safari, etc
- Firebug – http://getfirebug.com
- Dojo Firebug extensions

Notes:

- Unfortunately everything won't go perfectly for you during development, so you will need to debug your web pages.
- The easiest approach will be to avoid problems by using editors for your HTML, CSS, JavaScript, and Dojo. IBM's Rational Developer for System z (RDz), Rational Application Developer (RAD), and Eclipse with the Web Development Tools plugin have good editors. There are many editors available.
- Firebug consistently receives high ratings as a debugger for web pages

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.

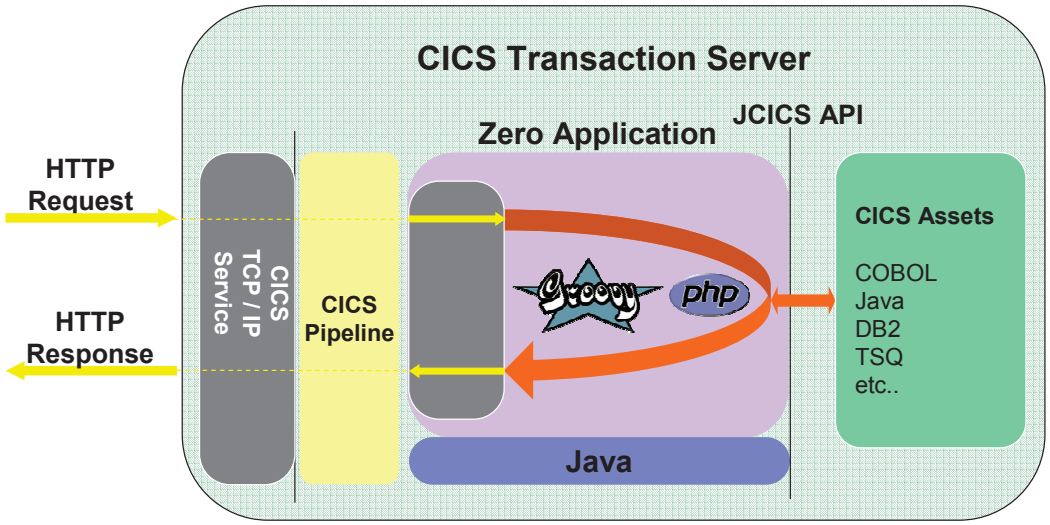
CICS Dynamic Scripting Feature Pack

- Technology from **Project Zero**, WebSphere sMash (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 **JVMServer** Technology
- 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** Feature Pack for CICS TS V4.1, June 2010
- Optional **no charge** Feature Pack for CICS TS V4.2, September 2011

Notes:

- CICS Dynamic Scripting is a Feature Pack for CICS TS 4.1 and CICS TS V4.2.
- 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 you will need to 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)
- **Relevant CICS Dynamic Scripting concepts:**
  - **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
- To implement a dependency, you would, after adding the dependency to your application's config/ivy.xml file, stop your application, do a zero resolve or a zero update, then a zero start to start your application.
- The 'zero resolve' leaves existing dependencies at their current levels. The 'zero update' implements the dependency at the highest available level.

Resolving Applications…

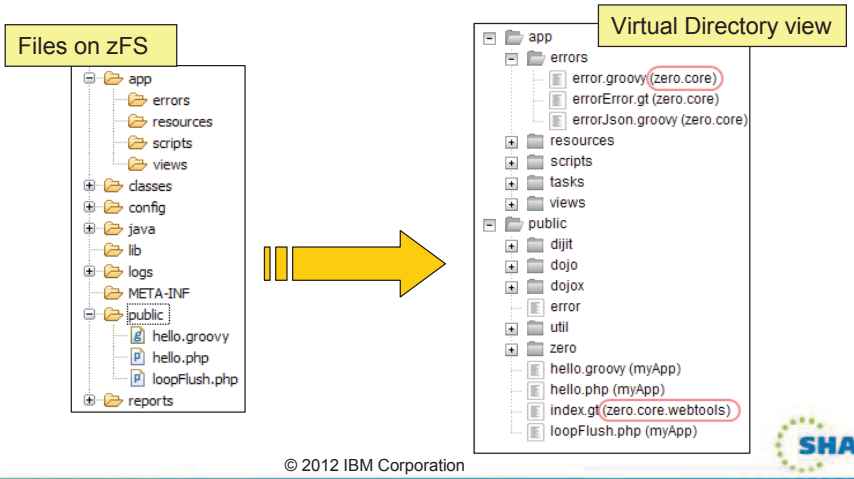
- **An application must be "resolved" before the it can be used. Resolving an app means:**
  - Locating its dependencies & determining exactly which versions to use.
  - Possibly retrieving them from a remote repository, if they are not found in the app's "workspace" or the CLI's local repository.
-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

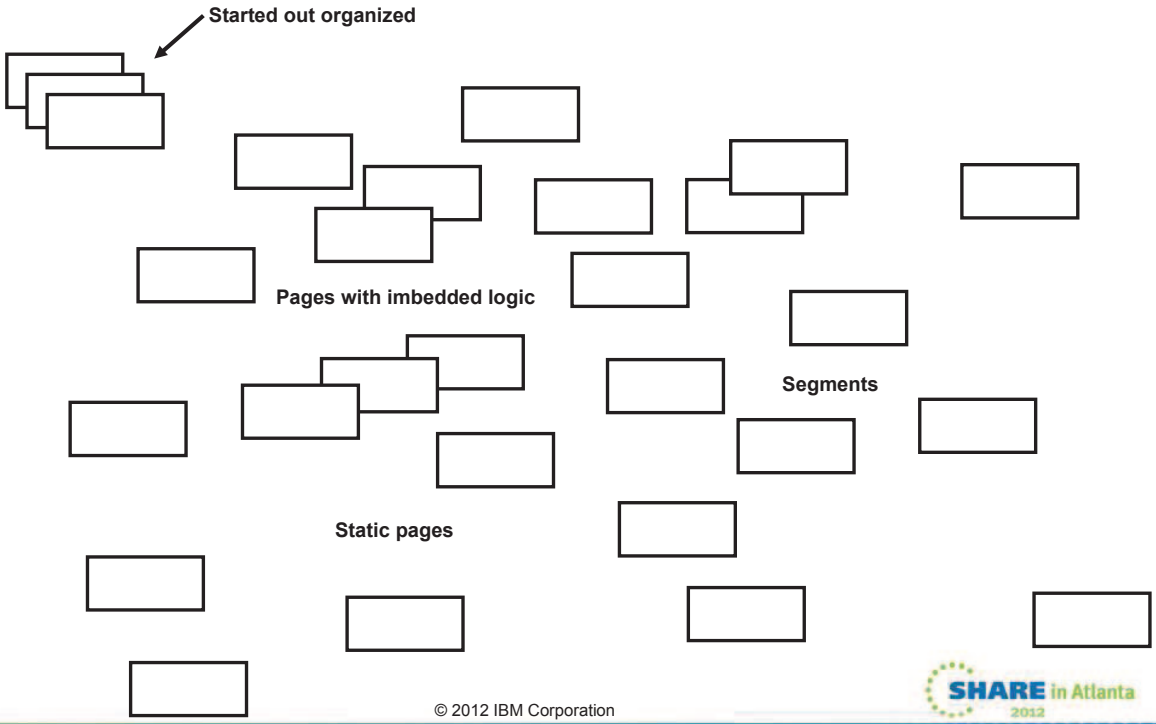
- **From the application developer's perspective, artifacts are "inherited" from dependencies.**
- **They are available through the concept of Virtualized Directories.**
  - The Virtualized Directory browser tool illustrates this. It can be added to any app by adding a dependency on the module zero.core.webtools.



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.

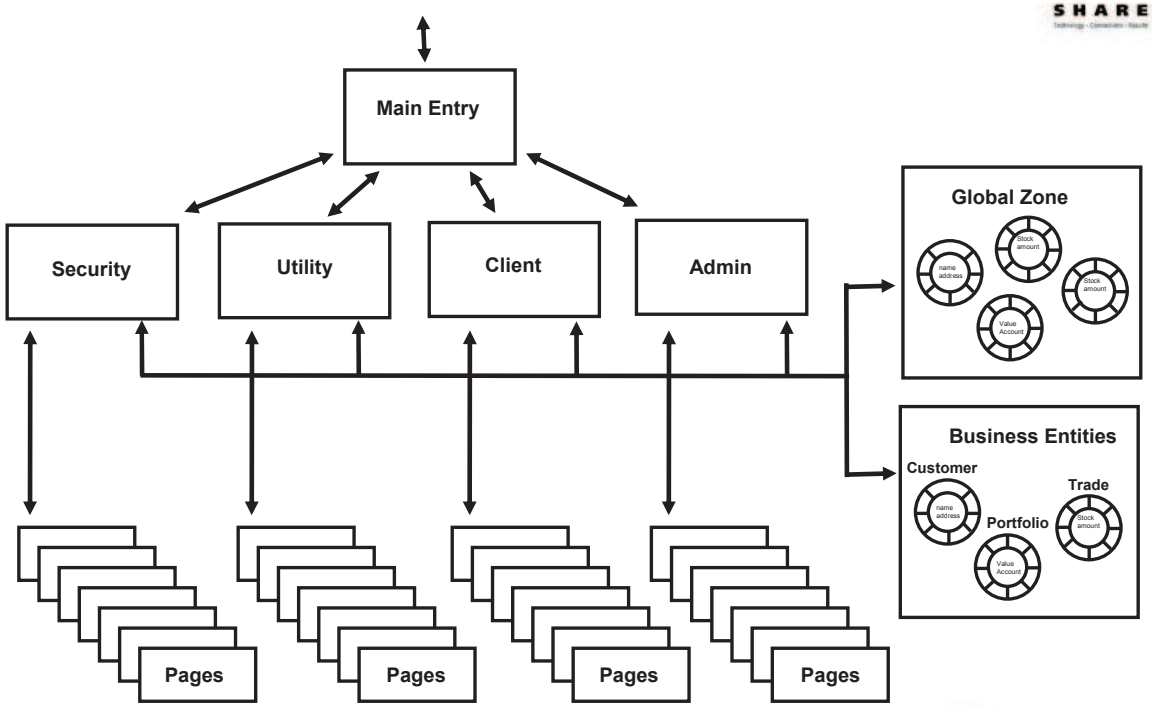
Typical PHP Application that got out of control



Notes:

- Before we start talking about the mechanics behind building web-based applications, we need to talk about application design.
- The CICS Dynamic Scripting environment provides a very flexible model for delivering web pages. It is therefore, very tempting to take 'the easy way out', and implement your application the very fastest way possible at that time. For example, you can if you wish, combine business logic and presentation logic in the same PHP script. This is obviously a poor choice since the business logic is not reusable, the program is way more complex than needed, and maintenance cost will go up exponentially. But, you say, it is only a small program and I need to get it done quickly.
- Most applications start small and grow. If a poor application design was taken initially, it is likely that as you add patches and new functionality to the system it will become more difficult to maintain, and therefore, more costly.
- CICS Dynamic Scripting offers several features that allow you to take a well organized approach to your application architecture.
- This slide is intended to illustrate a PHP-based application, that, although it's first few pages were somewhat organized is now out of control and costly to maintain. Due to the complexity, perhaps there is only one person at our shop that understands the application and can maintain it.

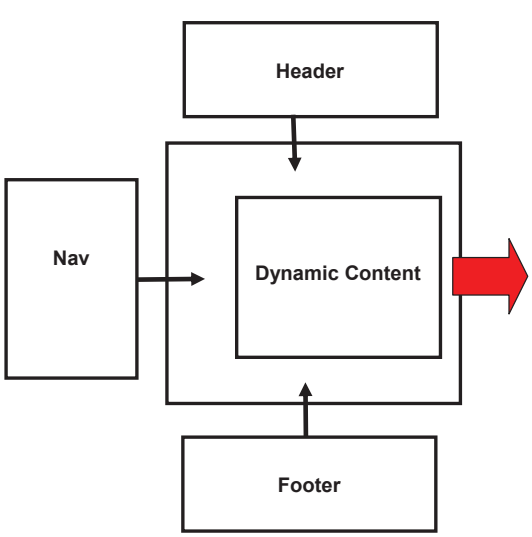
Nicely Structured Application….



Notes:

- This slide is intended to show an application that is well organized. This provides a framework existing logic can be quickly found and changed. New functions can easily be added to the framework.
- Additionally, due to the divisions, many application programmers can be working on the same application at the same time, and don't have to understand the entire application.
- A request from any web browser enters the main controller which delegates the request to the appropriate area. The Security controller can handle logon, customer information updates, registration, and password updates. The client controller, for our Fantasy Investing application, provides access to the client's portfolio, stock trades, loans, etc
- The controllers invoke business logic to implement the desired actions.
- The controllers may need to temporarily store information about our interactions with the user, so we can conduct what appears to be a 'conversation' with our user, although the interactions are discreet information exchanges.
- When the business actions are complete, the controller pass data to display logic that develops a web page or a REST response.
- CICS Dynamic Scripting has facilities that make this approach easy to implement.

Single web page composed of parts

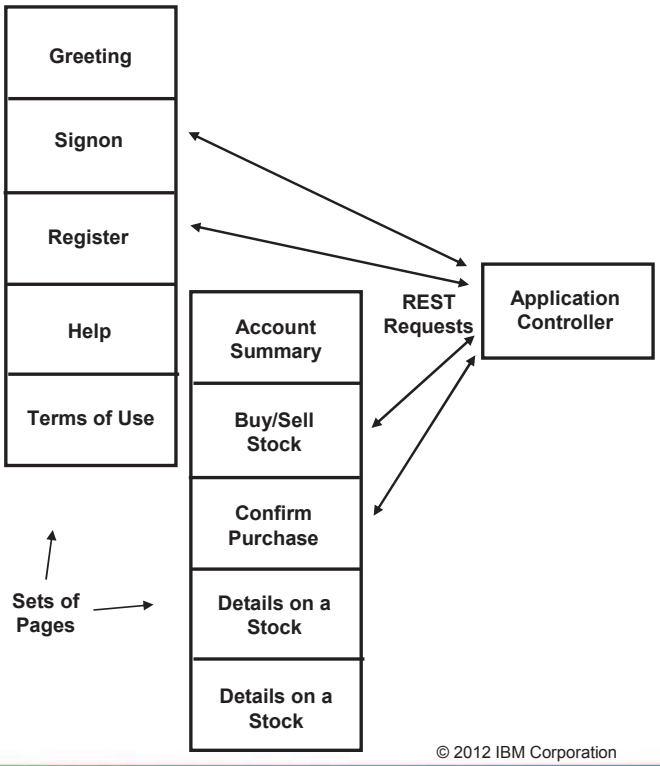


** Drawing from "Getting Started with IBM WebSphere sMash" book

Notes:

- Within a web page itself, we can further divide responsibilities.
- It is likely that the banner and footer of our application will be the same or very similar for most all pages within our application.
- Likewise the menu options can often be written in common, reusable code
- If we reuse common code for the banner, footer, and menus, then the application programmer only needs to address the dynamic part of the web page.
- Again, CICS Dynamic Scripting has facilities that make this approach easy to implement.

Mobile Applications….

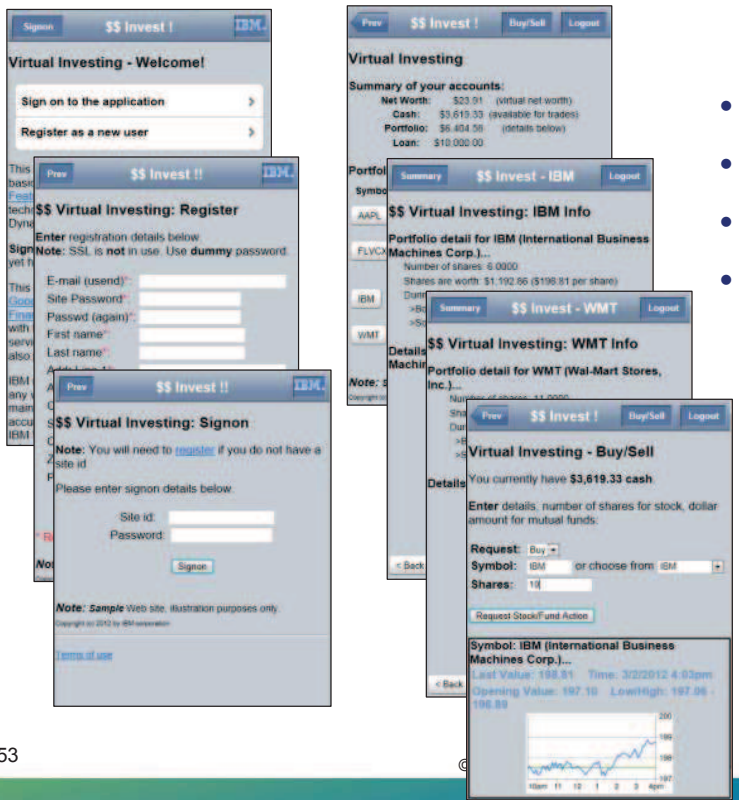


- For mobile applications, due to device speed and communications speed, it is typical to send a set of ‘views’ in a single HTML page
- The user navigates through the set of views, and the view makes a light-weight REST request to implement the business request
- Dojo makes this approach easy

Notes:

- Due to the speed of the mobile devices, and the speed off connectivity for mobile devices, you would normally have several “views” sent to a mobile device in the same web page.
- The user can quickly (since the views are already on the device) navigate between views.
- When a business function is requested, JavaScript routines make lightweight AJAX requests to the backend system.
- This provides for the best mobile experience
- Dojo can help in this area. We will see this on upcoming slides

Mobile Applications….



- Two HTML pages
- Multiple views per page
- Dojo formatted
- REST requests for business actions

dojo toolkit

Notes:

- This slide is an example of what you can do with Dojo
- Shown here are two HTML pages, each with multiple ‘views’
- When the page is delivered to the mobile web browser, the various views (each defined in a tag), are hidden (except for one).
- When a business action is needed, JavaScript on the HTML page makes an AJAX request and displays the results

Mobile Web Page with Dojo (1 of 2)

```
<html><head>
  <meta name="viewport" content="width=device-width,initial-
scale=1,maximum-scale=1,minimum-scale=1,user-scalable=no"/>
  <meta name="apple-mobile-web-app-capable" content="yes" />
  <title>Welcome</title>
 <link
href="http://ajax.googleapis.com/ajax/libs/dojo/1.7.1/dojox/mobile/t
hemes/iphone/iphone.css" rel="stylesheet" />
 <script type="text/javascript"
 src="http://ajax.googleapis.com/ajax/libs/dojo/1.7.1/dojo/dojo.js"
 djConfig="parseOnLoad:true"></script>
<script type="text/javascript">
  dojo.require("dojox.mobile.parser");
  dojo.require("dojox.mobile");
  dojo.require("dojox.mobile.Button");
  dojo.requireIf(!dojo.isWebKit,"dojox.mobile.compat");
</script>
</head><body style="visibility:hidden;">

 <!-- ***** multiple divs (views) go here - see next page ***** -->

</body></html>
```


Mobile Web Page with Dojo (2 of 2)

```
<!-- ***** example sigonView ***** -->
<div id="signonView" data-dojo-type="dojox.mobile.View" selected="true">
  <h1 data-dojo-type="dojox.mobile.Heading">$$ Invest !!
  <div data-dojo-type="dojox.mobile.ToolBarButton" style="float:left; "
   moveTo="initial" transitionDir="-1">Prev</div>
  <div data-dojo-type="dojox.mobile.ToolBarButton" icon="/graphics/id-ibm-
logo.gif" style="float:right;" href="http://m.ibm.com"></div>
</h1>
<small><h2>$$ Virtual Investing: Signon</h2></small>
<p>Please enter signon details below.</p>
<form name="theForm1"  method="post" action="/inv/scripts/INVSite.php"
       onsubmit=" return no1Entry(); ">
<input type="hidden" name="route" value="security" />
<input type="hidden" name="request" value="signMeOn" />
<center><table>
<tr><td>Site id:</td><td><input name="eMail" type="text" size="20"
       maxlength="75" value="" /></td></tr>
<tr><td>Password:</td><td><input name="password" type="password"
       size="20" maxlength="20" value="" /></td></tr>
</table>
<br/>
<input type="submit" value="Signon" />
</center>
</form>
</div>
```

Dynamic Scripting Locations for Artifacts

- **public Directory**
  - **Graphics, CSS, JavaScript**
- **app/scripts**
  - **Presentation Controller**
  - **Business logic (well, maybe just the call to the business logic)**
- **app/views**
  - **Pages and page fragments**
- **app/errors**
  - **Custom error messages**
- **State Data – (for a pseudo-conversational experience)**
  - Global Zone
  - Cookies
- **Internationalization Support**

Notes:

- There was a slide earlier in the deck that indicated that an “zero application” was a set of directories, each with their own purpose.
- This slide list some of the application directories that will be of interest when building presentation logic with CICS Dynamic Scripting.
- The public directory will hold your Cascading Style Sheets, JavaScript, and graphics
- The app/scripts directory will hold your controller logic, and may contain your business logic. If you have both of these in the app/scripts directory, then I recommend you place them in separate sub directories within the app/scripts. Your business logic might be implemented in regular CICS programs (e.g. COBOL), or in Java. In this case the app/scripts would only hold your controller logic.
- The app/views directory will hold web pages and web page fragments (more on this in a few slides)
- For anything but an extremely simple application, you will likely need to maintain state between user interactions. CICS Dynamic Scripting provides a ‘Global Zone’ for this (more on this in a few slides).
- The CICS Dynamic Scripting environment also provides for I18N.

Global Zone

- **Map of data**
- **Can access from PHP, Groovy, or Java**
- **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.
- The zones you will use most often in your programs are

  /request
  - Contains information about the input request, including form data, input and output headers, the URL, the path, the HTTP method and the query string.

  /user
  - Is used for maintaining session data

SHARE in Atlanta 2012

Receiving Requests

- **Form data**

```
$customerName = zget("/request/params/customerName");
```

- **Path**

```
$thePath = zget("/request/path");
```

- **Method**

```
$theMethod = zget("/request/method");
```

- **Query String**

```
zget("/request/queryString");
```

- **Headers**

```
zget("/request/headers/in/User-Agent");
zput("request/headers/out/Content-Type", "text/plain");
```


Notes:

- This side shows common zget and zput commands that you will use to get and put data from/to the Global zone

Session data

- /user zone = session data
  - Visible to all programs accessed for an HTTP request
  - Identified by the value of the zsessionid cookie value
  - Preserved across server recycles
  - Accessible from PHP, Groovy, or Java - Uses Java serialization, so only place serializable objects in it
  - An HTTP session times out after a period of inactivity
  - Timeout is set by /config/userZone/idleTimeout in zero.config
  - Session data is invalidated after zsessionid cookie expires, or manually
  - Default idle timeout is 5 minutes

```
zput("/user/someName", "value"); // store value or object
zget("/user/someName");          // get value or object
zpost("/user#invalidate",true);  // force invalidation
zpost("/user#save",true);    // force save to physical storage
```


Notes:

- The /user global zone is used for session data.
- If you want to maintain data between user interactions, you can place the data in the /user global zone
- CICS Dynamic Scripting keeps the session data for your session separate from others by associating it with a session id, and writing this session id to a cookie. You don't have to track session, you just use the /user zone

Providing Responses

- Static pages
- Write output to STDOUT (which is sent to the browser)
  - println (Groovy)
  - echo (PHP)
- Renderers
  - Place in app/views
  - PHP
    - It is an 'embedded' scripting language, so you can have HTML in your PHP file
  - Groovy
    - Groovy script (a .groovy file)
    - Groovy template (a .gt file, embedded)
      - *Similar to Java ServerPages*

Notes:

- When writing to the web browser, you can use the println command (Groovy) or echo (PHP)
- Renderers allow you to separate the web page generation from you controller logic
- The next few pages first show you an approach where you may be able to get pages developed quickly and may be ok for an application that has only a couple pages, however if that application ever expands to many web pages, this approach will quickly cause complications in the application since it is not well organized.

Simple PHP Pages

- **You will be tempted to mix business logic with your presentation logic; it is usually not a good idea**

```
<html>
<body>
<h1>A Title<h1>
<?php
// you could put database access (or anything else) here
echo "<ul>";
echo "<li>Can add dynamic content by invoking PHP functions ";
echo "and echoing the results!</li>";
echo "<li>You could even have business logic in the middle ";
echo "of your PHP pages (which is often tempting), but do not ";
echo "do this you want a clear division between the presentation ";
echo "and the business layer</li>";
echo "</ul>";
?>
</body>
</html>
```


Notes:

- Since you can mix PHP and HTML tags into a single file, you will often be tempted to create a your output web page, and then add PHP logic to the middle of the page that performs business functions.
- This is almost always a bad idea. If you take this approach and want to use that business logic again, you will have to duplicate that business logic in another script.
- Additionally, the person maintaining this script will have to know and understand both the presentation and business logic requirements of the application.
- It always a good idea to assume that any application you write is a worthwhile application. Since it is a worthwhile application, your application will grow to include many more functions, and will require maintenance.

PHP Renderer

- From the controller logic…

```
...
// invoke business logic
makeStockPurchase($client, $requestedStockSymbol, $requestedAmount);
// indicate data a page to be rendered
zput("/request/requestedStockSymbol", $requestedStockSymbol);
zput("/request/requestedAmount", $requestedAmount);
zput("/request/view", "inv/showStockPurchase.php");
render_view();
...
```

- In the showStockPurchase.php script (the view)

```
<?php
$requestedStockSymbol = zget("/request/requestedStockSymbol");
$requestedAmount = zget("/request/requestedAmount");
?>
<html><body><h1>Stock Trade Info</h1><hr/><br/>
You just purchased <?php echo $requestedAmount; ?>
shares of <?php echo $requestedStockSymbol; ?>.
<br/></body></html>
```


Notes:

- This slide shows the preferred approach
- The controller logic (top) invokes a business function to perform the desired action. The controller logic places objects returned from the business function into the /request zone, specifies the page to be displayed and invokes a renderer.
- The target page to be rendered gets the data objects from the /request zone and then generates the page to be sent to the web browser.

Page to be rendered – better example

- **Separate header/menu/trailer and main content**

```
<?php
 <!-- Get passed data -->
$requestedStockSymbol = zget("/request/requestedStockSymbol");
$requestedAmount = zget("/request/requestedAmount");
?>
<html>
 <!-- Header -->
<?php zput("/request/view", "inv/header.php"); render_view(); ?>
<!-- Dynamic Content -->
<h1>Stock Trade Info</h1>
You just purchased <?php echo $requestedAmount; ?>
shares of <?php echo $requestedStockSymbol; ?>.
<br/>
 <!-- Navigation and Footer -->
<?php zput("/request/view", "inv/nav_menu.php"); render_view(); ?>
<?php zput("/request/view", "inv/footer.php"); render_view(); ?>
</html>
```


Notes:

- This slide shows the preferred approach
- The controller logic (top) invokes a business function to perform the desired action. The controller logic places objects returned from the business function into the /request zone, specifies the page to be displayed and invokes a renderer.
- The target page to be rendered gets the data objects from the /request zone and then generates the page to be sent to the web browser.

Groovy

- The Groovy scripting language also available
- Can do the same as previously described with PHP (on the previous slides)
- Can have Groovy scripts (.groovy files)
- Can have Groovy templates (.gt files)
- Am not going to include Groovy code samples here…. See examples in:
  - "Getting Started with IBM WebSphere sMash" book
  - Project Zero documentation
  - WebSphere sMash documentation
  - CICS Dynamic Scripting Redbook

Notes:

- The same concepts and facilities for page rendering are also available using the Groovy scripting language. See the documentation in the reference section for examples.

File Browsing

- **Add the following to your zero.config:**

  /config/fileserver/directoryBrowsing = true

- **By default the /app/views/listing.gt from zero.core will be used to show directories**
  - Override with /config/fileserver/directoryView = “customDirectoryListing.gt”
- Not typically done in production
- If you have directories you don’t want people to access, add an index.html file to the directory

Notes:

- You can enable file browsing in application directories if you would like.
- This is usually not a good idea in a production system

Custom Error Messages

- Can use the default error message mechanism
- Can customize messages
- Can use your own error message mechanism
- Using the default error page is:

```
<?php
...
zput("/request/view", "error");
zput("/request/status", "500");
zput("/request/error/message", "You made an error");
render_view();
...

?>
```


Notes:

- CICS Dynamic Scripting comes with default error pages
- You can customize your own error pages

Security

- **Active Content Filtering**
  - **Removes potentially malicious active content from application content that is displayed in a browser**
  - **For example, an SQL injection attack**
  - `<dependency name="zero.acf" org="zero" rev="[1.0.0.0, 2.0.0.0["/>`
- **SSL**
  - **Specified in the config/zero.xml file**
  - **See the CICS documentation**
- **HTTP Basic Authentication**
  - **Specified in the config/zero.xml file**
  - **See the CICS documentation**

Notes:

- The CICS Dynamic Scripting environment provides ACF (Active Content Filtering) to guard against certain attacks such as an SQL injection attack
- CICS Dynamic Scripting applications can be protected with SSL or HTTP Basic Authentication. Since CICS is listening for incoming requests, you will want to read the CICS-related documentation on CICS Dynamic Scripting security
  - http://publib.boulder.ibm.com/infocenter/cicsts/v4r2/index.jsp?topic=%2Fcom.ibm.cics.ts.smash.doc%2Fsecuring_applications.html

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 kinds of 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.

CICS DS: REST: onList()

```
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.

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

Hello Dojo Introduces basic concepts of the Dojo JavaScript toolkit **Concepts:** Dojo, JavaScript, Dijit, widgets	**Connection API** 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** Demonstrates security features and illustrates how to leverage OpenID authentication **Concepts:** Open ID, authentication, security rules, extending a user registry
Suggestion Box 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** Illustrates how to render your data in Atom Syndication Format. **Concepts:**Atom	**IWidgets** Shows how to build and test IWidgets with IBM® WebSphere® sMash **Concepts:** IWidgets
Employee Data Provides an interface for managing a list of employees using RESTful conventions and SQL **Concepts:** SQL Data Access, REST, JSON	**Flow Samples** Demonstrate a few of the basic features of the flow language **Concepts:** Assemble Flow, feed processing	**Kicker and Receiver** 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 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** 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** Demonstrates some of the features of the Open AJAX 1.1 Hub provider (Secure mashup) **Concepts:** Open AJAX, mashups, security
	Flickr Server Side Mashup A simple Flickr-based mashup application developed using PHP **Concepts:** Flickr, mashups, PHP, Dojo	

Summary -- Session # 10291

- Web-related Terminology
  - HTTP, HTML, CSS, DOM, JavaScript
  - AJAX, Widgets, DOJO
  - Mobile
- What is CICS Dynamic Scripting (CICS DS)
- Web application design
  - The typical design I use
  - Maintaining state
- Getting CICS DS Request information
- CICS DS Response Rendering
  - Static web pages, CSS, JavaScript
  - Dynamic Web pages
    - PHP Rendering
    - Groovy Rendering
    - REST/AJAX
  - Error Responses
  - Implementing REST interfaces
- Security

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.