Enterprise Web Developer Workshop

Free Open Source Version

Rob Tweed

M/Gateway Developments Ltd

Introduction

- What is Enterprise Web Developer (EWD)?
  - A technology-agnostic framework for web application and Ajax development
- Why do people use it?
  - High productivity approach to web development
  - Very low maintenance
  - Creates modern Ajax applications
  - Automated security and session management

Free Open Source EWD

- Specifically targetted at GT.M as platform
- Only generates GT.M Mumps routines
- Only works with *m_apache* gateway

A Simple “Hello World”

- Ingredients:
  - – GT.M
  - – Apache
  - – *m_apache* gateway
  - – EWD installed and configured
  - – Your favourite editor
  - – Web browser

A Simple "Hello World"

- Create page as text file:

```
<ewd:config>
<html>
  <head>
    <title>EWD Workshop</title>
  </head>
  <body>
    <div>Hello World!</div>
  </body>
</html>
```

- Save as ***/usr/ewdapps/workshop/helloworld.ewd***

A Simple “Hello World”

- Compile:
  - do compileAll^%zewdAPI(“workshop”)
- Run:
  - http://192.168.1.123/ewd/workshop/helloworld.mgwsi

A Simple “Hello World”

- So how did all that work?
  - EWD Configuration
  - EWD Source Pages
  - EWD’s Compiler
  - *m_apache* Gateway

Install EWD

- Download:
  - Virtual Appliance
  - EWD routines and install in existing GT.M system

EWD Configuration

- Application Root Path
  - Where EWD source applications reside
  - /usr/ewdapps
  - Automatic configuration dialogue on first compilation
  - do setApplicationRootPath^%zewdAPI(“/usr/ewdapps”)
  - write $$getApplicationRootPath^%zewdAPI()
- Applications
  - /usr/ewdapps/workshop
  - /usr/ewdapps/finance
  - Just add new application subdirectories when needed

EWD Source Pages

- Simple text files
  - Reside in an EWD application subdirectory
  - eg */usr/ewdapps/workshop*
- File extension = ".ewd"
- Page names are up to you
- Recommended that they don't contain punctuation characters
- */usr/ewdapps/workshop/helloworld.ewd*
- */usr/ewdapps/finance/mainmenu.ewd*

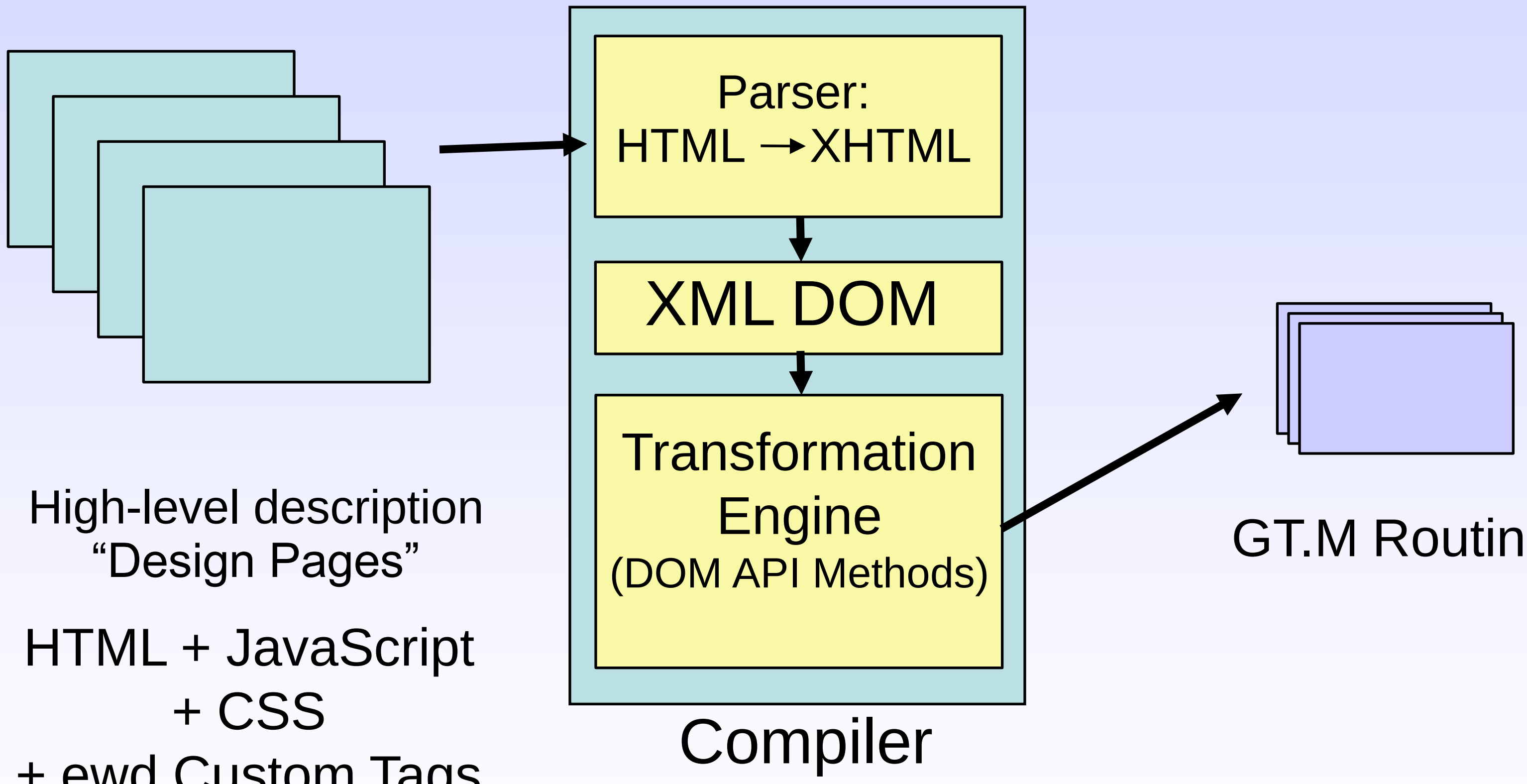
EWD Source Pages

- Mandatory <ewd:config> tag at start
  - Directives for EWD’s compiler
- Complete page:
  - HTML
  - Javascript
  - CSS styles
  - EWD custom tags
- In Ajax applications, can be a partial page or *“fragment”*

EWD’s Compiler

- Converts EWD source pages into run-time version:
  - GT.M routines
- Architecture:
  - Written in M code
  - HTML to XML converter/ DOM Parser
  - DOM transformation engine

EWD's Compiler



High-level description
"Design Pages"

HTML + JavaScript
+ CSS
+ ewd Custom Tags

EWD Configuration

- Output Routine Path
  - Destination path for compiled routines
  - *^zewd("config","routinePath","gtm")="/usr/local/gtm/ewd/"*

EWD’s Compiler

- Do compileAll^%zewdAPI(“workshop”)
  - Compile all the EWD pages in the application named “workshop”
    - Source pages will be found in /usr/ewdapps/workshop/*.ewd
  - Convert each .ewd file to a corresponding GT.M Mumps routine
    - Destination will be /usr/local/gtm/ewd
    - Routine name structure:
      - ewdWL[applicationName][pageName].m
      - eg ewdWLworkshopindex.m
  - You’ll find a few more files are generated automatically
    - Run-time support routines
    - Javascript files
    - CSS files

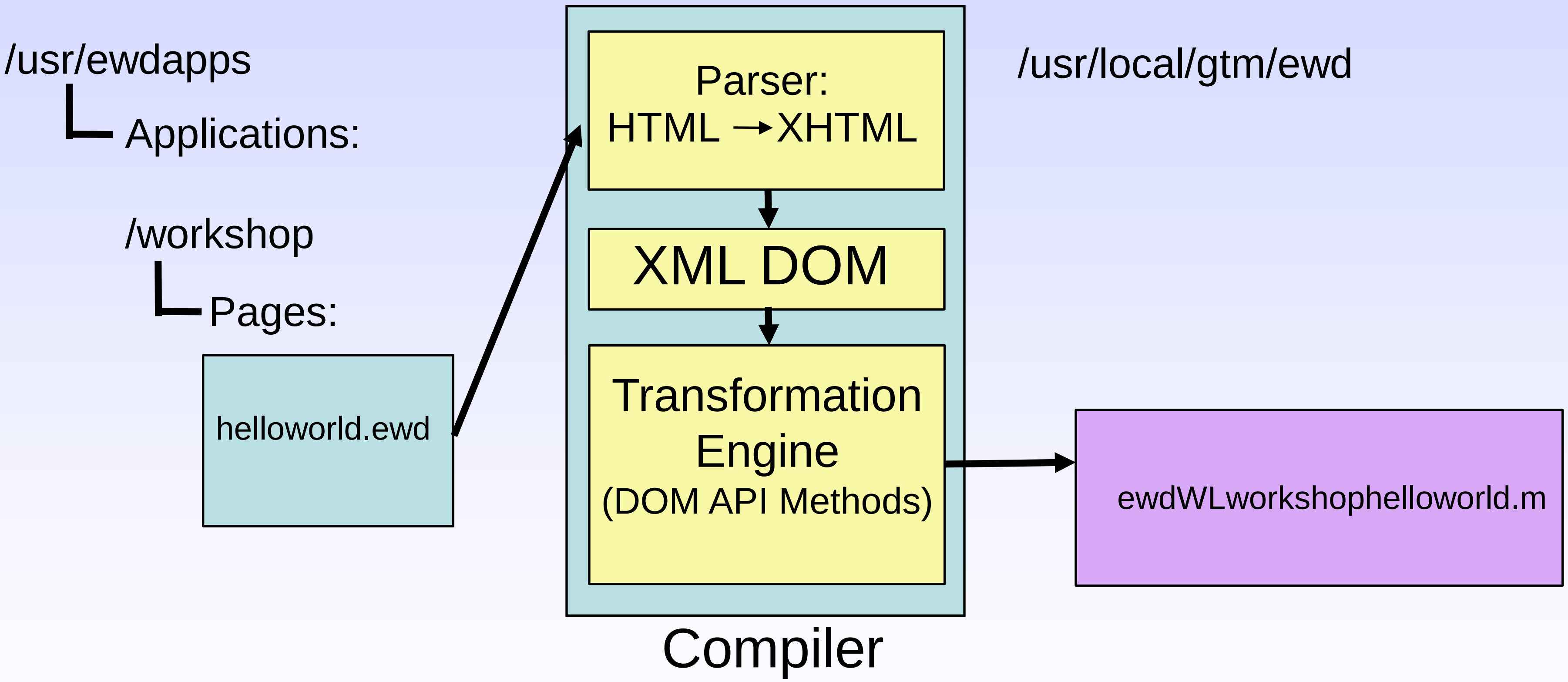
EWD's Compiler

Application Root Path:

/usr/ewdapps

Routine Path:

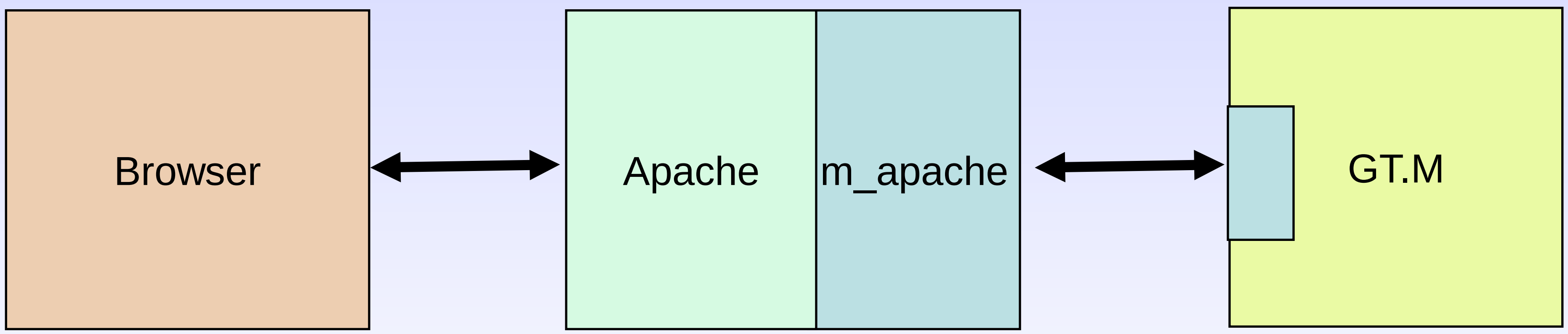
/usr/local/gtm/ewd



m_apache Gateway

- Open source Apache module
  - Very high performance
  - Highly scalable
- Connects Apache to GT.M system(s):
- Allows:
  - GT.M routines to be triggered via an HTTP request
  - – Remote access to GT.M globals and routines
  - – Send data into GT.M
  - – Send HTML pages from GT.M
  - – Can also be used for XML input/output and file uploads to GT.M over HTTP/HTTPS

m_apache Gateway



m_apache Gateway

- Pre-installed in EWD Virtual Appliance
- Included in EWD source file kit
- Install:
  - m_apache shared object file
    - Configure Apache to load it
  - GT.M daemon routines (^%ZMGWSI*)
    - Listen for incoming requests and action them

Managing *m_apache*

- Configured in Apache configuration file
- Started automatically by Apache
- To stop/restart: stop/restart Apache

Configuring *m_apache*

```
LoadModule m_apache_module /usr/mgwsi/bin/m_apache22.so
SetEnv MGWSI_PORT 7041

<Location /ewd>
  SetEnv MGWSI_M_UCI /usr/local/gtm/ewd/mumps.gld
  SetEnv MGWSI_M_FUNCTION runPage^%zewdGTMRuntime
</Location>

<Location "/mgwsi/sys/">
Order deny,allow
    Allow from all
</Location>
```

Invoking an EWD page

http://192.168.1.123/ewd/workshop/helloworld.mgwsi

Mapped m_apache URL path

Application name

EWD page name

Denotes this is an m_apache “file”

Hello World: So what?

- Wouldn’t it have been easier to just use a static page of HTML?
- What’s the significance of all this?

Dynamic generated pages

- The EWD framework that we’re now harnessing has all the ingredients you need:
  - To easily build dynamic, programmatically-generated pages and content
  - …in a secure and easy to maintain way

A more complex “Hello World”

- Create page as text file:

```
<ewd:config prePageScript=“helloWorld^ewdWorkshop”>
<html>
  <head>
    <title>EWD Workshop</title>
  </head>
  <body>
    <div><?= #helloWorld ?></div>
  </body>
</html>
```

- Save as ***/usr/ewdapps/workshop/helloworld2.ewd***

A more complex “Hello World”

- Create the script routine ^ewdDemo:

```
ewdDemo ; EWD Workshop scripts
 ;
helloWorld(sessid)
 d setSessionValue^%zewdAPI(“helloWorld”,”Hello world!”,sessid)
 QUIT “”
```

A more complex “Hello World”

- Compile and run the page:
  - d compilePage^%zewdAPI(“workshop”,”helloworld2”)
    - compileAll compiled all pages in the application
    - compilePage just compiles the specified page
    - First time you compile anything in a new application, use compileAll, then you can use compilePage thereafter
  - http://192.168.1.123/ewd/workshop/helloworld2.mgwsi

So what happened?

- EWD called out to a pre-page script
  - Allows you to fetch data from GT.M
  - Pre-page scripts have a single input (sessid)
  - Pre-page scripts always QUIT “”
- Our pre-page script created an EWD Session variable
  - Named helloWorld
- Our page was then able to access and display this variable:
  - <?= #helloWorld ?>

EWD Sessions

- A web application provides an illusion:
  - Ongoing meaningful dialogue between the browser and GT.M
- In fact it's a stateless environment
  - Separate, unconnected request/response pairs
- Session state is manufactured by EWD
  - During the session (eg from the user logging in until they log out), selected information is persisted
  - The persistent information is known as the EWD Session

EWD Sessions

- helloWorld2.ewd created a session
  - Look in the global ^%zewdSession(“session”)
  - Now forget about the global!
    - EWD provides access to the session using a wide range of API methods
  - EWD Session is a mixture of:
    - EWD-generated values
    - Programmer-generated values
    - Scalar values and sparse, multi-dimensional arrays

The ewdMgr Application

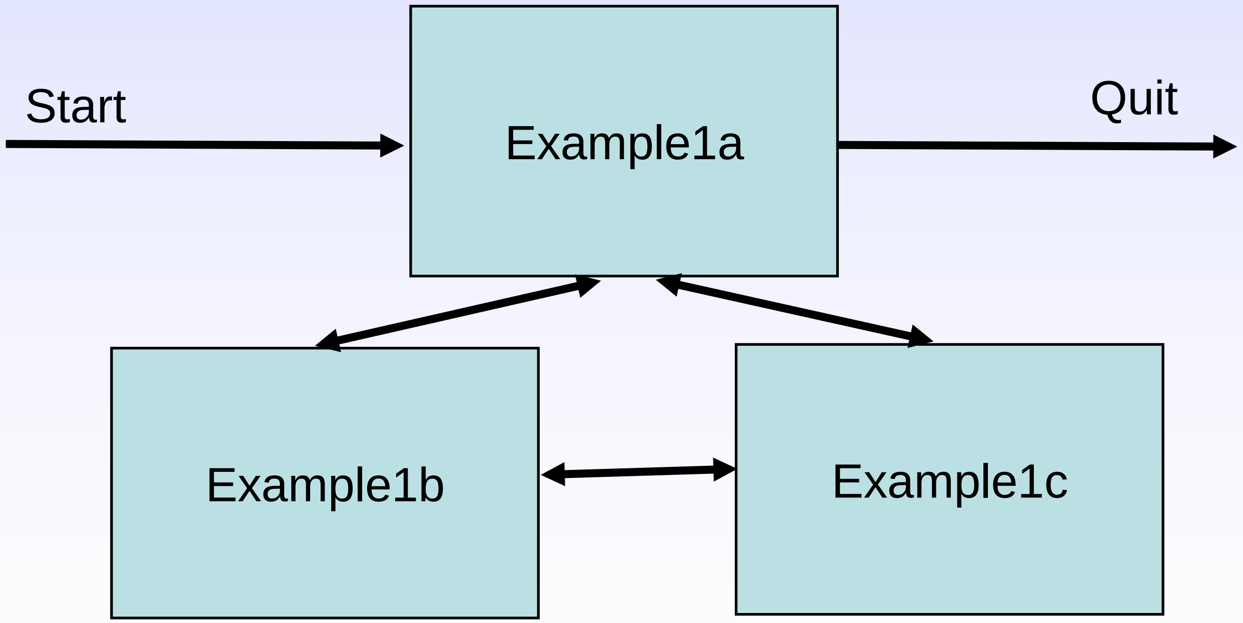
- EWD includes a web application named *ewdMgr*:
  - Manage and maintain your EWD environment
  - View Sessions

EWD Sessions

- EWD Session values are available:
  - In GT.M
  - In your EWD pages
  - EWD conceptually and logically separates these two:
    - Back-end
    - Front-end

Sessions & Persistence

- Let's create a simple application
- Three simple linked pages:



Session & Persistence

Example1a.ewd

```
<ewd:config isFirstPage="true">
<html>
  <head>
    <title>Session & Persistence</title>
  </head>
  <body>
    <div>This is the First page</div>
       <p><a href=“Example1b.ewd">Go to page 2</a></p>
       <p><a href=“Example1c.ewd">Go to page 3</a></p>
      <p><a href="ewdLogout.ewd">Log out</a></p>
  </body>
</html>
```

Example1b.ewd

```
<ewd:config isFirstPage=“false">
<html>
  <head>
    <title>Session & Persistence</title>
  </head>
  <body>
    <div>This is the Third page</div>
       <p><a href=“Example1a.ewd">Go to page 1</a></p>
       <p><a href=“Example1c.ewd">Go to page 3</a></p>
      <p><a href="ewdLogout.ewd">Log out</a></p>
  </body>
</html>
```

Example1c.ewd

```
<ewd:config isFirstPage=“false">
<html>
  <head>
    <title>Session & Persistence</title>
  </head>
  <body>
    <div>This is the Third page</div>
       <p><a href=“Example1a.ewd">Go to page 1</a></p>
       <p><a href=“Example1b.ewd">Go to page 2</a></p>
      <p><a href="ewdLogout.ewd">Log out</a></p>
  </body>
</html>
```

Create these files and add them to /usr/ewdapps/workshop/

Session & Persistence

- Compile the application again:
  - D compileAll^%zewdAPI(“workshop”)
- Run the first page:
  - *http://192.168.1.123/ewd/workshop/Example1a.mgwsi*
  - Examine the session using ewdMgr
- Click on a link
  - Session will still exist, but will have been updated
- Select logout
  - Session has been deleted by EWD

First and Other pages

- Try starting with another page, eg:
  - *http://192.168.1.123/ewd/workshop/Example1b.mgwsi*
  - – You should see an error!
- <ewd:config isFirstPage="true">
  - – Can be accessed via a simple URL
- <ewd:config isFirstPage="false">
  - – Only accessible with a tokenised URL
  - – Compare the <a> tags in the EWD page and the compiled routine

EWD tokens

- EWD automatically adds unique tokens to every link or form

```
<a href='/ewd/workshop/Example1a.mgwsi?ewd_token=mMUgDwDV0Su7biQgX3pQCyM3eDl9Zw
    &n=v8msTC07RMs03vhWT65CCsMO7RiJt6&ewd_urlNo=Example1b1'>
```

- All HTTP requests within a session are tokenised
  - *ewd_token* is a proxy to the Unique session ID
  - Randomly generated string, known only to the back-end
  - Nextpage token (*n*) prevents unauthorised, arbitrary access to pages
  - Try changing the page name in the Location window!

EWD Session

- Starts when someone invokes an un-tokenised URL to an EWD application's First Page
- Identified by unique Session ID (sessid)
  - Simple integer value
- Finishes when a user logs out:
  - Invokes a link to the pseudo-page *ewdLogout.ewd*
- Or times out:
  - 20 mins of no activity (can be reset)

A simple form

```
<ewd:config isFirstPage="true">
```

- This will be a First Page, so:
    - it will start a new session
    - It can be accessed with a simple, un-tokenised URL

A simple form

```
<ewd:config isFirstPage="true">
<html>
  <head>
    <title>EWD Workshop: Simple Form</title>
  </head>
  <body>
    <p>This is a simple EWD form</p>
    <h3>Please enter your username and password:</h3>
```

A simple form

```
<ewd:config isFirstPage="true">
<html>
  <head>
    <title>EWD Workshop: Simple Form</title>
  </head>
  <body>
    <p>This is a simple EWD form</p>
    <h3>Please enter your username and password:</h3>
    <form method="post" action="ewd">
```

- Always use action="ewd"
- Instructs EWD's compiler to apply its form automation

A simple form

```
<ewd:config isFirstPage="true">
<html>
  <head>
    <title>EWD Workshop: Simple Form</title>
  </head>
  <body>
    <p>This is a simple EWD form</p>
    <h3>Please enter your username and password:</h3>
    <form method="post" action="ewd">
      <table border=0>
        <tr>
          <td>Username: </td>
            <td><input type="text" name="username" focus="true" value="*"></td>
        </tr>
```

- **Standard text field:**
  - **Must have name or id defined**
  - **Value should be set to "*"**
  - **focus="true" will automatically set focus to field**

A simple form

```
<form method="post" action="ewd">
  <table border=0>
    <tr>
      <td>Username: </td>
        <td><input type="text" name="username" focus="true" value="*"></td>
    </tr>
    <tr>
        <td>Password: </td>
        <td><input type="password" name="password"></td>
    </tr>
```

A simple form

```
<form method="post" action="ewd">
  <table border=0>
    <tr>
      <td>Username: </td>
        <td><input type="text" name="username" focus="true" value="*"></td>
    </tr>
    <tr>
        <td>Password: </td>
        <td><input type="password" name="password"></td>
    </tr>
    <tr>
        <td>
           <input type="submit" name="Submit" value="Login"
                action="login^ewdWorkshop" nextpage="Example2b">
        </td>
    </tr>
```

A simple form

```
<form method="post" action="ewd">
  <table border=0>
    <tr>
      <td>Username: </td>
        <td><input type="text" name="username" focus="true" value="*"></td>
    </tr>
    <tr>
        <td>Password: </td>
        <td><input type="password" name="password"></td>
    </tr>
    <tr>
        <td>
           <input type="submit" name="Submit" value="Login"
                action="login^ewdWorkshop" nextpage="Example2b">
        </td>
    </tr>
```

A simple form

```
<ewd:config isFirstPage="true">

<html>
 <head>
  <title>EWD Workshop: Simple Form</title>
 </head>
 <body>
  <p>This is a simple EWD form</p>
  <h3>Please enter your username and password:</h3>
  <form method="post" action="ewd">
   <table border=0>
    <tr>
     <td>Username: </td>
      <td><input type="text" name="username" focus="true" value="*"></td>
    </tr>
    <tr>
      <td>Password: </td>
      <td><input type="password" name="password"></td>
    </tr>
    <tr>
      <td><input type="submit" name="Submit" value="Login" action="login^ewdWorkshop" nextpage="Page2d"></td>
    </tr>
   </table>
  </form>
  <br><br>
  <a href="ewdLogout.ewd">Log out</a>
 </body>
</html>
```

A simple form

- Save as /usr/ewdapps/workshop/Example2a.ewd
- Compile:
  - Do compilePage^%zewdAPI(“workshop”, ”Example2a”)
- Before we run it, what about the action script:
  - **login^ewdWorkshop**

Action scripts

- Invoked when a form is submitted or some other event triggered
  - validate posted data
  - Manipulate posted data
  - Save data
- A form may have multiple submit buttons:
  - Assign a different action script to each one

Action scripts

- Extrinsic function
- Single parameter:
  - sessid
  - Automatically passed by EWD at run-time

Action scripts

```
login(sessid)
 ; ….etc
 QUIT error
```

- An action script will always return a string value
  - Indicates success or failure of script
  - Null value = success
  - Non-null value = error message

Action scripts

```
login(sessid)
 n username
 s username=$$getSessionValue^%zewdAPI("username",sessid)
```

Action scripts

```
login(sessid)
  n username
  s username=$$getSessionValue^%zewdAPI("username",sessid)
```

```
<input type="text" name="username" focus="true" value="*”>
```

Action scripts

```
login(sessid)
  n username
  s username=$$getSessionValue^%zewdAPI("username",sessid)
```

```
<input type="text" name="username" focus="true" value="*”>
```

- **EWD automatically maps text fields to Session Values**
- **Value assigned when form submitted**
- **Named according to field name/id**
- **Accessed in Action Script using $$getSessionValue^%zewdAPI()**

Action scripts

```
login(sessid)
  n password,username
  s username=$$getSessionValue^%zewdAPI("username",sessid)
  s password=$$getSessionValue^%zewdAPI("password",sessid)
```

- Password fields treated exactly like text fields

Action scripts

```
login(sessid)
  n password,username
  s username=$$getSessionValue^%zewdAPI("username",sessid)
  s password=$$getSessionValue^%zewdAPI("password",sessid)
  i username'="ROB" QUIT “Invalid username"
  i password'="ROB" QUIT “Invalid password"
  QUIT ""
```

Validate the submitted fields and return any errors

Action scripts

```
login(sessid)
  n password,username
  s username=$$getSessionValue^%zewdAPI("username",sessid)
  s password=$$getSessionValue^%zewdAPI("password",sessid)
  i username'="ROB" QUIT “Invalid username"
  i password'="ROB" QUIT “Invalid password"
  QUIT ""
```

Save ewdWorkshop.m

A simple form

- What about the next page:

```
<input type="submit" name="Submit" value="Login"
              action="login^ewdWorkshop" nextpage="Example2b">
```

Let’s create it….

A simple form

Example2b.ewd:

```
<ewd:config isFirstPage="false">
```

A simple form

```
<ewd:config isFirstPage="false">

<html>
  <head>
    <title>Workshop: Simple form</title>
  </head>
  <body>
    <div>This is the second page</div>
    <br>Hello <?= #username ?><br>
    Your password is <?= #password ?><br>
    Your session ID is <?= #ewd.sessid ?>
    <br><br>
    <a href="ewdLogout.ewd">Log out</a>
    <hr>
  </body>
</html>
```

A simple form

```
<ewd:config isFirstPage="false">

<html>
  <head>
    <title>Workshop: Simple form</title>
  </head>
  <body>
    <div>This is the second page</div>
    <br>Hello <?= #username ?> <br>
    Your password is <?= #password ?><br>
    Your session ID is <?= #ewd.sessid ?>
    <br><br>
    <a href="ewdLogout.ewd">Log out</a>
    <hr>
  </body>
</html>
```

Syntax for accessing a Session variable within a page
# denotes Session variable

A simple form

```
<ewd:config isFirstPage="false" mgwsiServer="ewd">

<html>
  <head>
    <title>Workshop: Simple form</title>
  </head>
  <body>
    <div>This is the second page</div>
    <br>Hello <?= #username ?> <br>
    Your password is <?= #password ?><br>
    Your session ID is <?= #ewd.sessid ?>
    <br><br>
    <a href="ewdLogout.ewd">Log out</a>
    <hr>
  </body>
</html>
```

- **Session object**
- **In this case, automatically created by EWD**

A simple form

```
<ewd:config isFirstPage="false">

<html>
  <head>
    <title>Workshop: Simple form</title>
  </head>
  <body>
    <div>This is the second page</div>
    <br>Hello <?= #username ?><br>
    Your password is <?= #password ?><br>
    Your session ID is <?= #ewd.sessid ?>
    <br><br>
    <a href="ewdLogout.ewd">Log out</a>
    <hr>
  </body>
</html>
```

- **Save as Example2b.ewd**

A simple form

- Compile Example2b.ewd:
  - Do compilePage^%zewdAPI(“workshop”,”Example2b”)
- Run the form example:

  http://192.168.1.123/ewd/workshop/Example2a.mgwsi

A simple form

- Pre-assigning values to forms
  - – Fetching values from database
  - – Displaying those values in form fields
- Pre-page script
  - – Used to fetch data for the page
  - – Let’s add one to our form example:
    - Pre-assign a value for the username

Pre-page Scripts

```
<ewd:config isFirstPage="true“ prePageScript=“getUsername^ewdWorkshop” >
<html>
  <head>
    <title>EWD Workshop: Simple Form</title>
  </head>
  <body>
    <p>This is a simple EWD form</p>
    <h3>Please enter your username and password:</h3>
    <form method="post" action="ewd">
      <table border=0>
        <tr>
          <td>Username: </td>
            <td><input type="text" name="username" focus="true" value="*"></td>
        </tr>
```

Pre-page Scripts

```
getUsername(sessid)
 QUIT “”
```

- A pre-page script always has a single input parameter:
  - *sessid*
  - Automatically passed at run-time by EWD
- A pre-page script will always return a string value
  - Indicates success or failure of script
  - Null value = success
  - Non-null value = error message
  - Usually null !

Pre-page Scripts

```
getUsername(sessid)
  d setSessionValue^%zewdAPI(“username”,”Rob”,sessid)
  QUIT “”
```

Pre-page Scripts

```
getUsername(sessid)
   d setSessionValue^%zewdAPI(“username”,”Rob”,sessid)
   QUIT “”
```

```
<input type="text" name="username" focus="true" value="*”>
```

Pre-page Scripts

```
getUsername(sessid)
   d setSessionValue^%zewdAPI(“username”,”Rob”,sessid)
   QUIT “”
```

```
<input type="text" name="username" focus="true" value="*”>
```

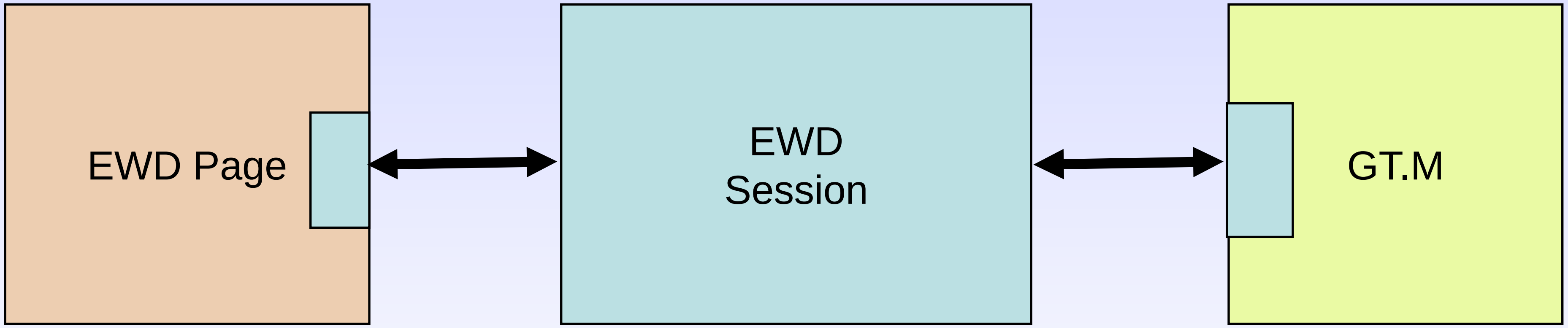
By setting a Session value, EWD will display the value in the matching form field

Compile and re-run and see for yourself!

Accessing the Session

- Creating Session values:
  - At the back-end (ie in Cache):
    - Pre-page scripts
      - setSessionValue^%zewdAPI
  - Automatically, eg when forms submitted
- Accessing Session values:
  - Within the page
    - <?= #name ?>
    - Form fields (mapped by name)
  - Within action and pre-page scripts:
    - getSessionValue^%zewdAPI

The Session as Interface



Typical form

- See example at http://www.mgateway.com
  - Simple6Page1.ewd

Typical form

- <ewd:comment>
  - Example of an EWD custom tag
  - EWD’s compiler removes this tag and any contents from the page
  - Allows comments in source page only

Session “Containers”

- HTML form field types:
  - Text and Password fields
  - Hidden fields
  - Radio buttons
  - Checkboxes
  - Select (drop-down menus)
    - Single choice
    - Multi-choice
  - Textarea

Hidden Fields

- Treated the same as text and password fields
- Single value
- Mapped to/from simple Session value

Radio Buttons

```
<tr>
       <td>User Type: </td>
        <td>
                Administrator : <input type="radio" name="userType" value="a">
                Pro User : <input type="radio" name="userType" value="p">
                Guest : <input type="radio" name="userType" value="g">
        </td>
</tr>
```

Note: nothing that says how a radio button is checked

Radio Buttons

- Pre-page script:
  - do setSessionValue^%zewdAPI(“userType”,”p”,sessid)
- Action script:
  - s userType=$$getSessionValue^%zewdAPI(“userType”,sessid)

- Single-value item
- Controlled by a simple session value

Single-option Select

<select name=“userType”></select>

Note: nothing that describes what the <option> values are, or which is to be pre-selected

Single-option Select

- Pre-page script:
  - d clearList^%zewdAPI(“userType”,sessid)
  - d appendToList^%zewdAPI(“userType”,”Administrator”,”a”,sessid)
  - d appendToList^%zewdAPI(“userType”,”Pro User”,”p”,sessid)
  - d setSessionValue^%zewdAPI(“userType”,”p”,sessid)
- Action script:
  - s userType=$$getSessionValue^%zewdAPI(“userType”,sessid)

- Single-value item
- Controlled by a simple session value

Checkboxes

```
<tr>
	<td>User Permissions: </td>
	<td>
		Web access : <input type="checkbox" id="permissions" value="w">
		Email : <input type="checkbox" id="permissions" value="e">
		Scheduler : <input type="checkbox" id="permissions" value="s">
	</td>
</tr>
```

Note: nothing that says which checkboxes are checked or how

Checkboxes

- Pre-page script:

d initialiseCheckbox^%zewdAPI("permissions",sessid)

d setCheckboxOn^%zewdAPI("permissions","w",sessid)

d setCheckboxOn^%zewdAPI("permissions","e",sessid)

- Action script:
  - d getCheckboxValues^%zewdAPI("permissions",.selected,sessid)
    - selected(“w”)=“w”
  - $$isCheckboxOn^%zewdAPI(“permissions”,”e”,sessid)
  - $$isSelected^%zewdAPI(“permissions”,”e”,sessid)

- Multi-value item
- Controlled by session array

Textarea

```
 <tr>
           <td>Comments: </td>
           <td><textarea id="comments" rows=20 cols=100>*</textarea></td>
</tr>
```

Note: nothing that says how textarea is pre-populated

Textarea

- Pre-page script:

```
d clearTextArea^%zewdAPI("comments",sessid)
s textarea(1)="This is a line of text”
s textarea(2)="This is the second line“
d createTextArea^%zewdAPI("comments",.textarea,sessid)
```

- Action script:
  - d getTextArea^%zewdAPI("comments",.comments,sessid)
    - comments(lineNo) = text

“What” versus “How”

- EWD describes what your page will do
- Not how it will do it
- Pages are simple and easy to understand
- Simple and easy to maintain
  - Remember: you may not be the person maintaining the page in the future!

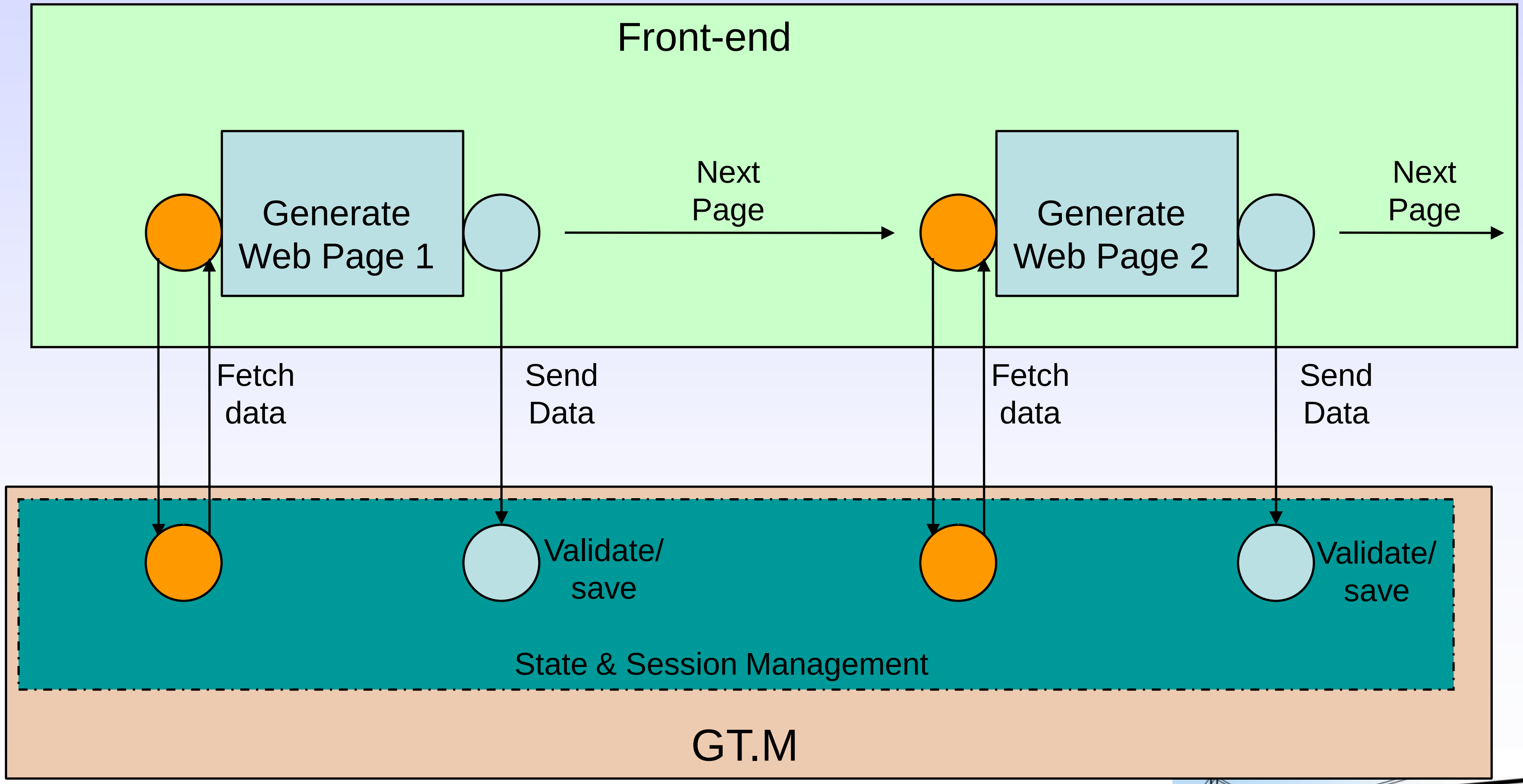
No Data Binding

- EWD does not use data binding
  - One of the biggest problems in other development tools
  - Page design not constrained by data model
- Uses implicit mapping between page and back-end via Session “containers” to provide de-coupled interface
  - Page designer doesn’t need to know where the data comes from or how
  - Programmer doesn’t need to know anything about how the data has been used in the page

Model View Controller?

- A very different approach to the MVC
  - URL mapped to method mapped to page content
- Focus is on the page, not URLs
  - In EWD you think how pages are linked
    - Event (described within the page)
    - Results in: Action script (method)
    - And takes you to: Next page
  - URLs are generated for you and actual linkages are automatically invoked

The EWD Event Model



Design versus Programming

- EWD focuses on page design
- Programming is reduced to:
  - Pre-page scripts:
    - Fetch data for use in the page
  - Action scripts:
    - Validate and save data
- Everything else is automated

Storyboard

- EWD allows you to treat a web application much like a static web site
  - Create a storyboard of initially unscripted pages
- And why not?
  - Only difference is that in a web application:
    - Some of the content you see is fetched from database or programmatically generated
    - Route taken through the pages may be determined by database and/or data entered by user
- Web applications are 90% design
  - Why are they treated as a programming task?

Ajax

- Breaks away from the page replace model of the “classic” web application
- Just replace parts of the page within the browser when an event occurs
- See example at http://www.mgateway.com
  - – ajaxContainer1.ewd

EWD’s Ajax Framework

- Container page
  - Initial complete HTML page that populates the browser
- Fragments
  - EWD pages that represent just chunks of markup
  - Treated otherwise as EWD pages
    - Pre-page script
    - Custom tags

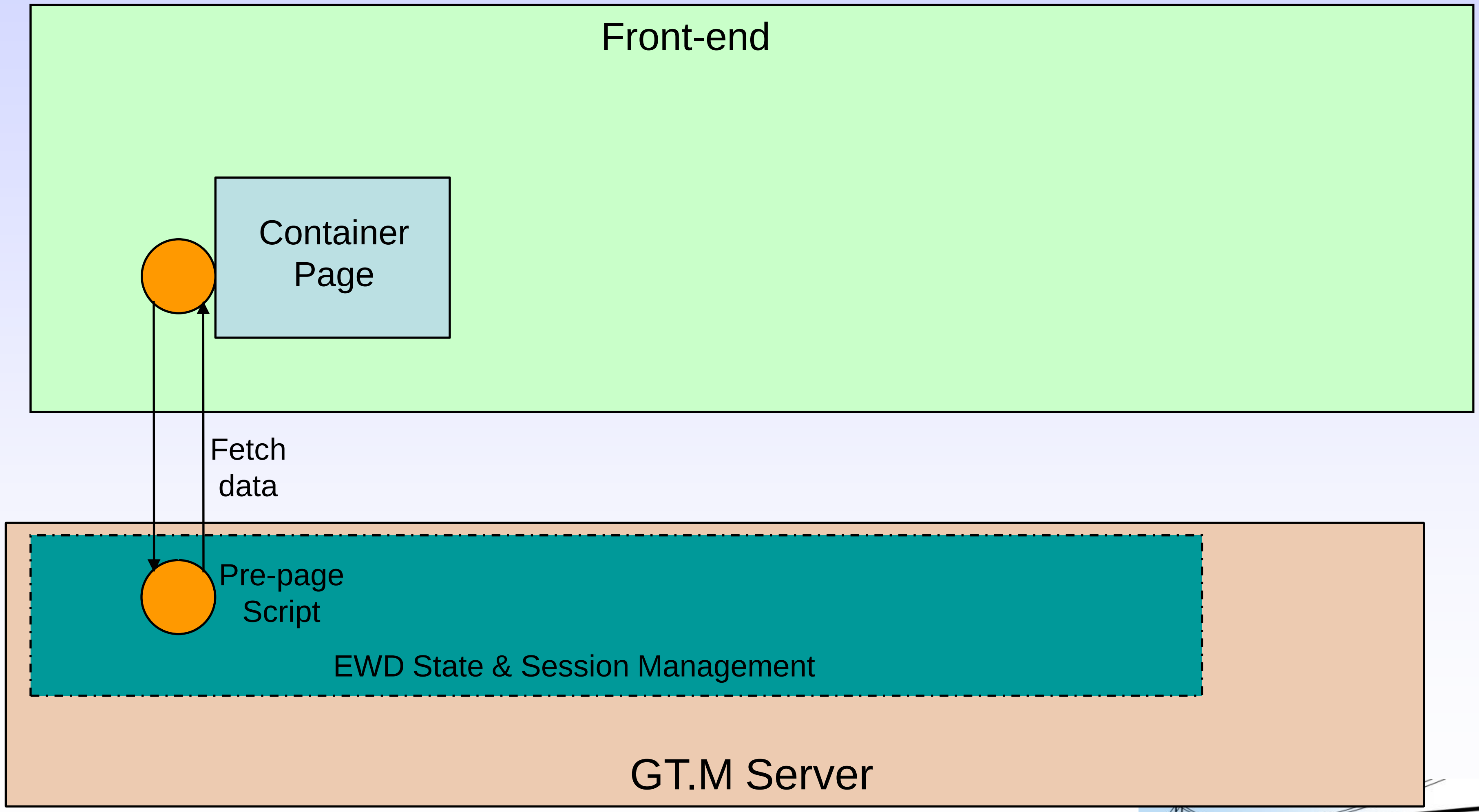
EWD’s Ajax Framework

- Event
  - Eg onClick
- Fetches Page fragment
- Replaces innerHTML of target tag (targetId)

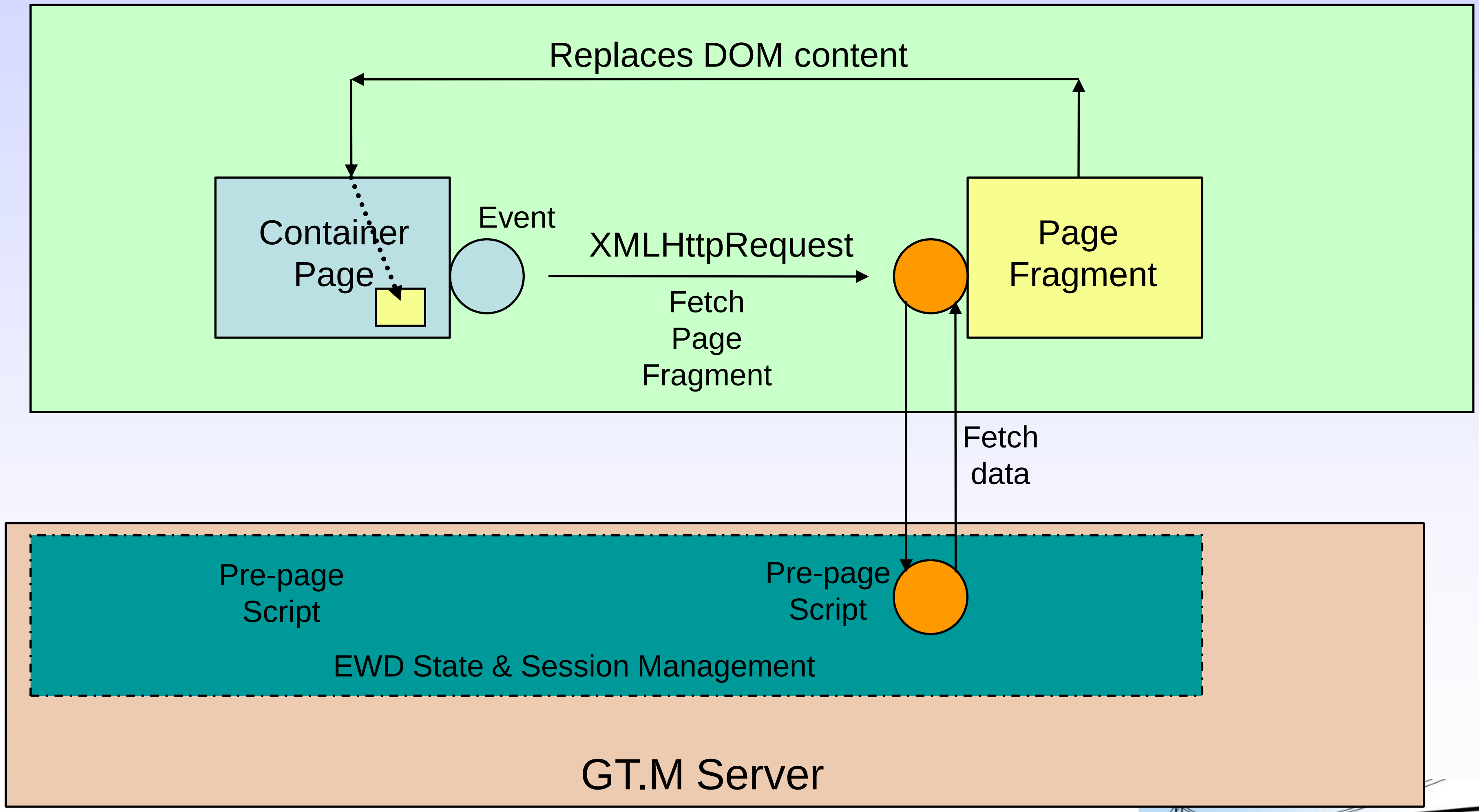
EWD Page Fragments

- <ewd:config pageType="ajax">
- No <html>, <head>, <body> tag
- Usually an outer <span> tag
- Fragments can deliver more target ids
  - Pages can grow in size and complexity as events occur through user interaction
  - Break down a complex page into many small fragments
    - Ease of maintenance
    - Team development

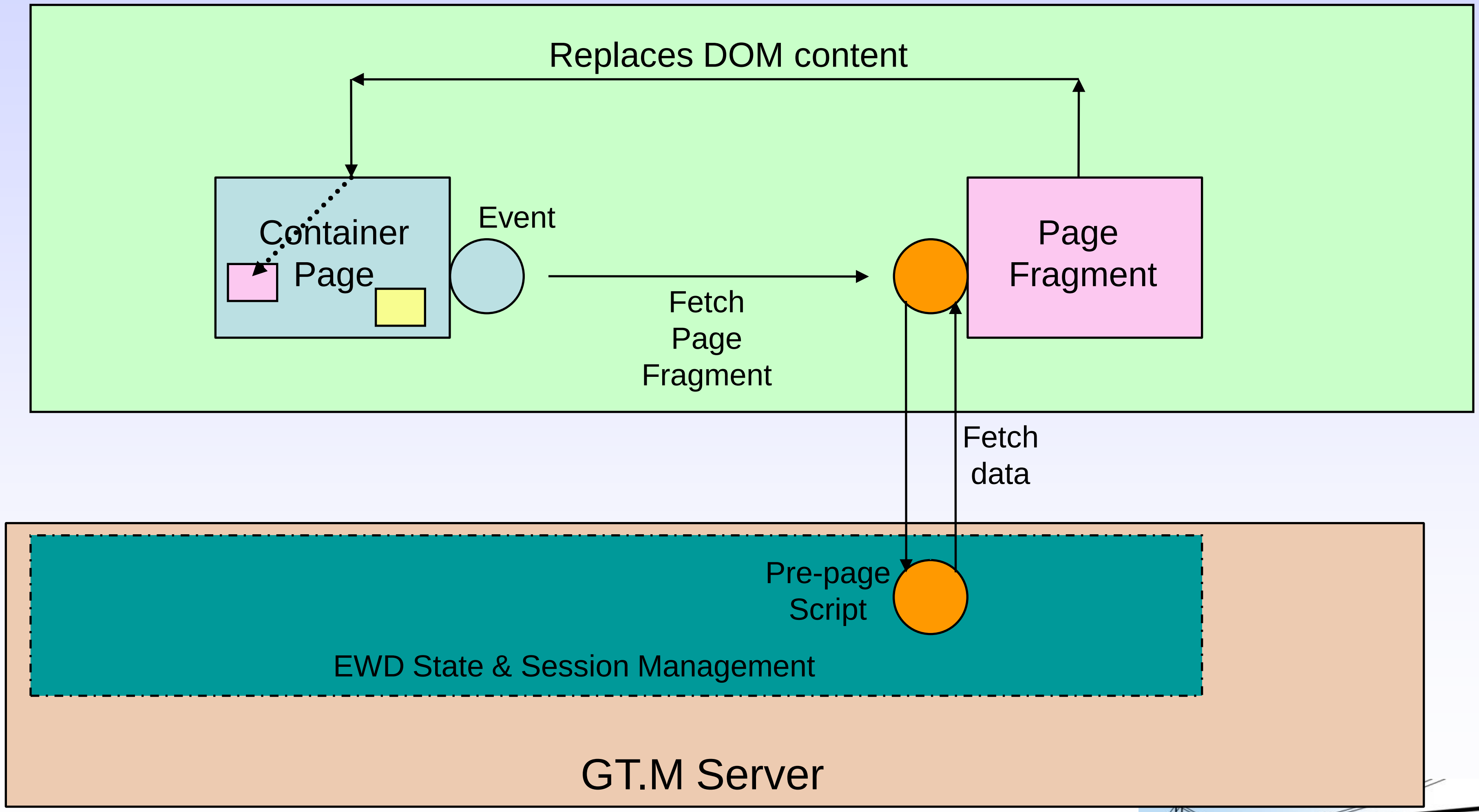
EWD: Ajax Application Flow



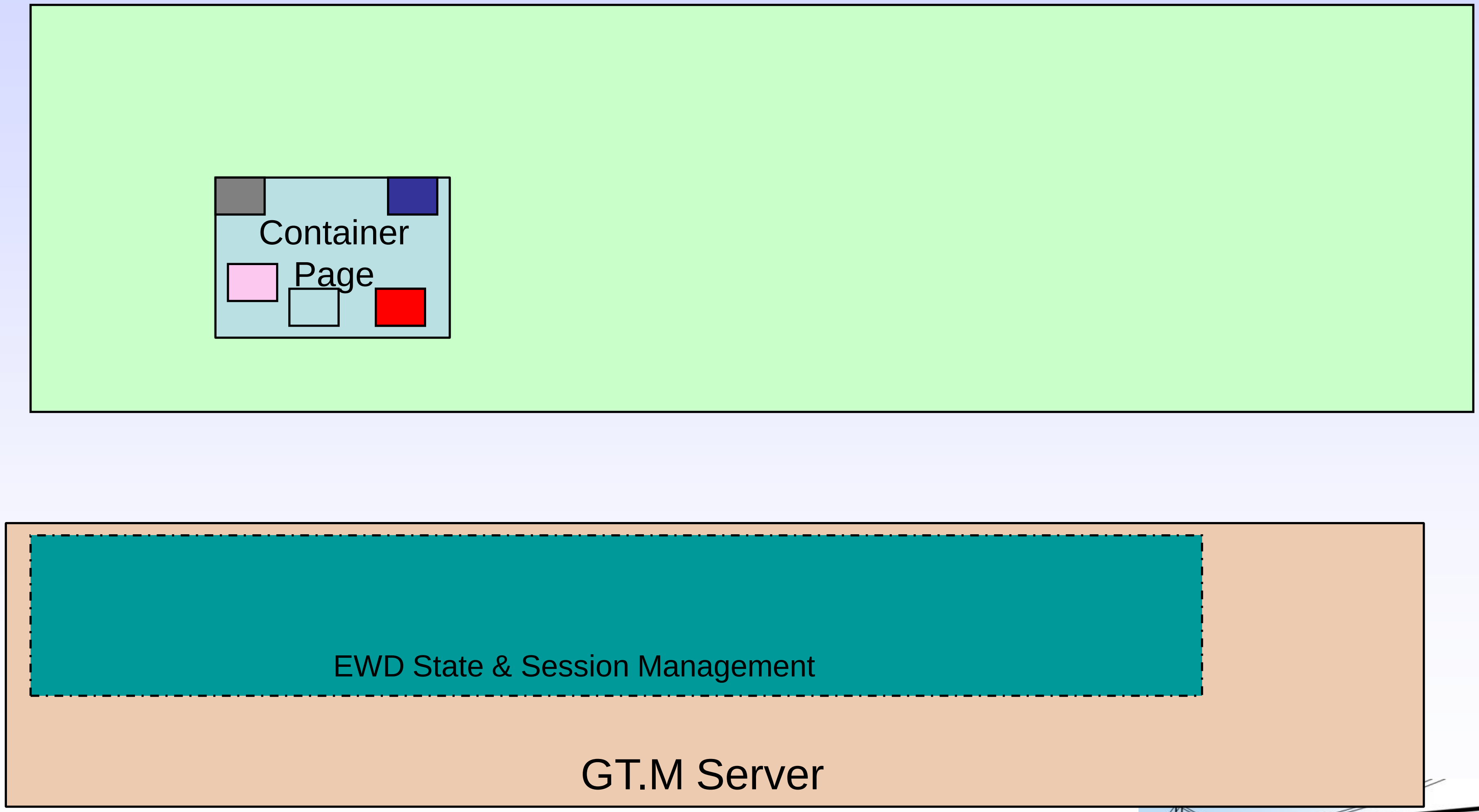
EWD: Ajax Application Flow



EWD: Ajax Application Flow



EWD: Ajax Application Flow



Manipulating the Session

- Scalar values:
  - *do setSessionValue^%zewdAPI(name,value,sessid)*
  - *do setSessionValue^%zewdAPI("a1","hello",sessid)*

  - *$$getSessionValue^%zewdAPI(name,sessid)*
  - *set a1=$$getSessionValue^%zewdAPI("a1",sessid)*

  - *do deleteFromSession^%zewdAPI(name,sessid)*
  - *do deleteFromSession^%zewdAPI("a1",sessid)*

Manipulating the Session

- Arrays
  - Sparse, multi-dimensional
  - Usual technique is to merge to and from a local array

```
set arr(“x”,1)=“value1”
set arr(“x”,2)=“value2”
set arr(“y”,10)=“another value”
do deleteFromSession^%zewdAPI(“arrx”,sessid)
do mergeArrayToSession^%zewdAPI(.arr,”arrx”,sessid)

kill myArray
do mergeArrayFromSession^%zewdAPI(.myArray,”arrx”,sessid)
```

In-page scripting

- Use EWD custom tags
- EWD can cross-compile these to any supported technology
- Conditions
- Loops
- Assignments
- Comments

In-page scripting

- Conditions
  - <ewd:if>
  - <ewd:ifBrowser>
  - <ewd:ifContains>
  - <ewd:ifSessionArrayExists>
  - <ewd:ifSessionNameExists>
  - All conditional tags can use:
    - <ewd:else>
    - <ewd:elseif>

In-page scripting: conditions

- All markup inside a conditional tag is rendered if the bounding condition is true at run-time

```
<ewd:if firstValue="#myValue" operation="=" secondValue="2">
   <div>Hello world</div>
</ewd:if>
```

“Hello world” will only be displayed if the session variable *myValue* equals 2

In-page scripting: conditions

- <ewd:if>
  - firstValue= literal, local or session variable
  - operation= “=“, “!=“ “<“ “>”
  - secondValue literal, local or session variable

```
<ewd:if firstValue=“#myValue” operation=“=“ secondValue=“2”>
   <div>Hello world</div>
</ewd:if>
```

In-page scripting: conditions

- <ewd:else>
  - No attributes

```
<ewd:if firstValue="#myValue" operation="=" secondValue="2">
   <div>Hello world</div>
<ewd:else>
   <a href="page2.ewd>Click here</a>
</ewd:if>
```

In-page scripting: conditions

- <ewd:elseif>
  - Same attributes as <ewd:if>

```
<ewd:if firstValue="#myValue" operation="=" secondValue="2">
   <div>Hello world</div>
<ewd:elseif firstValue="#myValue" operation="=" secondValue="3">
 <div>Hello world</div>
<ewd:else>
   <a href="page2.ewd>Click here</a>
</ewd:if>
```

In-page scripting: conditions

- Other useful conditional tags:
  - <ewd:ifBrowser type="ie7">
  - <ewd:ifBrowser os="xp">
  - <ewd:ifSessionNameExists sessionName=“myVar”>
  - <ewd:ifContains input="$data" substring=".ewd">

In-page scripting

- Loops
  - <ewd:for>
  - <ewd:forEach>

In-page scripting: loops

- <ewd:for>

```
<ewd:for from=“1” to=“10” increment=“1” counter=“$i”>
  <div>This is iteration number <?= $i ?></div>
</ewd:for>
```

In-page scripting: loops

- <ewd:forEach>
  - Allows you to iterate through a specified subscript level within a session array
  - Typically used to create data-driven table rows
  - Much like $order
  - Can be nested as deep as required
  - Attributes:
    - *sessionName: the name of the session array*
    - *index: the value of the iteration subscript*
    - *paramn: fixed values of previous subscripts*
    - *return: the data value of the array at the current iteration*

In-page scripting: loops

```
myArray(“account1”,”office1”)=“London Head Office”
myArray(“account1”,”office2”)=“London North”
myArray(“account2”,”office1”)=“New York Head Office”
```

```
<ewd:forEach sessionName=“myArray” index=“$ac”>
   <div>Account: <?= $ac ?></div>
</ewd:forEach>
```

```
Account: account1
Account: account2
```

M/GATEWAY DEVELOPMENTS

In-page scripting: loops

```
myArray("account1","office1")="London Head Office"
myArray("account1","office2")="London North"
myArray("account2","office1")="New York Head Office"
```

```
<ewd:forEach sessionName="myArray" index="$ac">
   <div>Account: <?= $ac ?></div>
  <ewd:forEach sessionName="myArray" param1="$ac" index="$office" return="$data">
   <div>Office: <?= $data ?></div>
  </ewd:forEach>
</ewd:forEach>
```

```
Account: account1
Office: London Head Office
Office: London North
Account: account2
Office: New York Head Office
```

In-page scripting

- Assignments
  - <ewd:set>
  - <ewd:getPiece>
  - <ewd:incrementCounter>
  - <ewd:modulo>
  - <ewd:replace>

In-page scripts: assignments

- <ewd:set>

```
<ewd:set return=“$var1” value=“100”>

<ewd:set return=“$var2” firstValue=“$var1" operand="+" secondValue="3">
```

operands: any valid Cache operand (+ - * / \ _)

In-page scripts: assignments

- <ewd:getPiece>

```
<ewd:getPiece return="$var" data="$input" delimiter="^" pieceNumber=“2”>

<ewd:getPiece return="$var" data="$input" asciiDelimiter="1" pieceNumber=“2”>
```

In-page scripts: assignments

- <ewd:incrementCounter>
  - <ewd:incrementCounter return="$counter">
- <ewd:modulo>
  - <ewd:modulo return="$value" data="$input" modulus="2">
- <ewd:replace>
  - <ewd:replace input="$data" fromString="xx" toString="yy" return="$newdata">

In-page scripts

- Comments
  - <ewd:comment>
  - Anything between opening and closing tag is ignored by the compiler
  - Allows you to put comments into your source EWD pages that will never be visible to a user

---

```
<ewd:comment>
  This will be ignored by the compiler
</ewd:comment>
```

In-page scripts

- Comments within Javascript
  - Can use standard Javascript comments
    - But these can be seen by the user
  - Can’t use <ewd:comment> tags within Javascript
    - Entire Javascript block inside <script> tag is processed by compiler as an XML text node
    - <ewd:comment> would be treated as text

In-page scripts: comments

- Comments within Javascript:
  - /*ewd:comment and */ewd:comment
  - All text inside is ignored by compiler

```
<script language="javascript>
  alert("You'll see this!") ;
  /*ewd:comment
     This will be ignored
 */ewd:comment
 alert("You'll see this too!") ;
</script>
```

In-page scripts: comments

- Comments within Javascript:
  - /*ewd:comment and */ewd:comment
  - All text inside is ignored by compiler

```
<script language=“javascript>
  alert(“You’ll see this!”) ;
  alert(“You’ll see this too!”) ;
</script>
```

In-page scripts

- EWD provides a comprehensive XML-based language for in-page scripting
- Intended to be used sparingly!
  - Remember: EWD page describes what, not how!
  - All too easy to obscure page functionality with lots of EWD custom tags
  - Perform as much of your data transformation logic in Cache in your pre-page script
  - Session variables and arrays should contain pre-processed values that can be simply dropped into the page

In-page scripts: KISS

Compare this….

```
<ewd:forEach sessionName=“myData” index=“$recNo” return=“$text”>
   <ewd:modulo return=“$isOdd” data=“$recNo” modulus=“2” />
   <ewd:if firstValue=“$isOdd” operation=“=“ secondValue=“0”>
      <ewd:set return=“$class” value=“blackBackground”>
   <ewd:else>
     <ewd:set return=“$class” value=“whiteBackground”>
   </ewd:if>
   <div class=“<?= $class ?>”><?= $text ?></div>
</ewd:forEach>
```

In-page scripts: KISS

with this….

```
<ewd:forEach sessionName="myData" index="$recNo" return="$data>
   <ewd:getPiece return="$text" delimiter="^" pieceNo="1">
   <ewd:getPiece return="$class" delimiter="^" pieceNo="2">
   <div class="<?= $class ?>"><?= $text ?></div>
</ewd:forEach>
```

- Pre-page script pre-calculated the modulo and assigned the class, then stored it in the session array data.
- Much easier to see what the page is doing

Session variable limitations

- Scalar session variables can easily become a random assortment of name/value pairs
  - Difficult to manage and maintain
  - Risk of over-writing a value you otherwise want to persist
  - or wrongly using a value that happens to already exist
- Session objects can be a better approach

Session Objects

- <?= #person.name ?>
- <?= #person.dob ?>
- <?= #person.address.zipcode ?>

Session Objects

- Creating and accessing session objects in your pre-page script:

```
do setSessionValue^%zewdAPI(“person.name”,”Rob Tweed”,sessid)

do setSessionValue^%zewdAPI(“person.address.zipCode”,”12345”,sessid)


set zip=$$getSessionValue^%zewdAPI(“person.address.zipCode”,sessid)
```

Session Arrays

- EWD Session arrays can carry complex data payloads:

```
set record(1)=“Rob^Tweed^UK”
set record(2)=“David^Rapperport^USA”
do mergeArrayToSession^%zewdAPI(“info”,.record,sessid)

<ewd:forEach sessionName=“info” index=“$no” return=“$data”>
   <ewd:getPiece return=“$firstname” delimiter=“^” pieceNo=“1”>
   <ewd:getPiece return=“$surname” delimiter=“^” pieceNo=“2”>
   <ewd:getPiece return=“$country” delimiter=“^” pieceNo=“3”>
   <div><?= $firstname ?> <?= surname ?> : Country= <?= $country ?></div>
</ewd:forEach>
```

Session Array Limitations

- Downside of this approach is that the data structure is arbitrary
  - not self-documenting
  - Maintenance over time becomes difficult
  - Modifications must be made with care
- Session resultSets are a better alternative
- Can manually create Session resultSet records

Session ResultSets: Manual

```
set propsArray(“name”)=”Rob”
set propsArray(“city”)=”London”
set propsArray(“country”)=”UK”
do mergeRecordArrayToResultSet^%zewdAPI(“info”,.propsArray,sessid)
…repeat for each record
```

```
<ewd:forEach sessionName=“info" index=“$recNo">
  <tr>
  <td><?= #info[$recNo].name ?></td>
  <td><?= #info[$recNo].city ?></td>
  <td><?= #info[$recNo].country ?></td>
  </tr>
</ewd:forEach>
```

Session ResultSets

- Deleting and clearing down:

```
Do deleteFromSession^%zewdAPI(“info”,sessid)
```

Managing your session

- Good idea to keep the session contents under control
  - Keeping track of names
- Use prefixes for your session names
  - do clearSessionByPrefix^%zewdAPI(prefix,sessid)
- Temporary session variables

Temporary Session Variables

- Prefix: *tmp.*
  - *tmp.myVar1*
  - *tmp.myVar2*
- Create in your pre-page Script
  - ***do setSessionValue^%zewdAPI("tmp.myVar1","hello",sessid)***
- Available to the current page
  - <?= #tmp.myVar1 ?>
- Not persisted – no maintenance required
- eg use for page of results for display only

Session versus local variables

- Session variables:
  - Communication between front- and back-end
  - Persistent information between pages
- Local variables (eg *$no*):
  - Locally scoped to current page only
  - No persistence between pages
  - Most useful within <ewd:forEach> loops