
ORACLE®

Best Practices for Delivering Framework Products that include Ajax Features

Peter Laird

Oracle Safe Harbor

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Your Speaker

- Architect for WebCenter and Portal products at Oracle
- 8 years on WebLogic Portal at BEA, acquired by Oracle in 2008

Session Goals

- Explain Oracle's experience in the area of delivering products with Ajax and JavaScript capabilities
- Deliver a set of best practices
- Cover the Case Studies that support those best practices

Target Audience

- You develop a product that is used by web application developers, and
- You have/plan to introduce Ajax/JavaScript features into your product, and
- You support web application developers doing custom development on top of your product.

Defining Terms

- There are several types of JavaScript solutions
  - Toolkit
  - Framework
  - Library
  - API
- These best practices generally apply to all
  - So we won't be precise during this talk
- Same goes for…
  - Ajax vs. JavaScript vs. DHTML

Agenda

- Introducing Oracle's Case Studies
- Framework Selection Best Practices
  - Open Source vs. Custom
- Framework Features Best Practices
  - Client side JavaScript API
  - Client side eventing
  - Enhancing the XmlHttpRequest
- Related Best Practices
  - RESTful data APIs
  - Single Origin Policy

Oracle's Case Studies

- Oracle provides Ajax and JavaScript enabled development platforms for different skill sets
- Case Studies
  - ADF Faces Rich Client
  - Application Express (APEX)
  - WebLogic Portal (WLP)
  - AquaLogic* Open Controls

* AquaLogic was a BEA brand, no longer in use

ADF Faces Rich Client

- What it is
  - Ajax enabled JSF component library
- Skill sets
  - Java developers
  - 4GL developers (Oracle Forms, VB, PowerBuilder)
- Brief History
  - UIX
  - ADF Faces
  - Apache Trinidad
  - ADF Faces Rich Components

Application Express (APEX)

- What it is
  - A rapid web app development tool
  - Serves pages from out of Oracle database (!)
- Skill set
  - PL/SQL
- Brief History
  - HTML DB
  - Ajax introduced as an internal feature

WebLogic Portal

- What is it
  - Enterprise portal platform
- Skill set
  - Java developers
- Brief History
  - Java rendering framework
  - Ajax and JavaScript added much later

AquaLogic Open Controls

- What it is
  - JavaScript enabled JSF component library
  - Internal to BEA, for use by new BEA products
  - Has a unique feature to support .NET products
- Skill sets
  - Java developers
  - .NET developers

Agenda

- Introducing Oracle Case Studies
- **Framework Selection Best Practices**
  - **Open Source vs. Custom**
- Framework Features Best Practices
  - Client side JavaScript API
  - Client side eventing
  - Enhancing the XmlHttpRequest
- Related Best Practices
  - RESTful data APIs
  - Single Origin Policy

Open Source or Custom?

- Difficult to justify creating a custom Ajax/JavaScript framework anymore
- The open source frameworks provide sizable feature sets
- But there are niche cases where a custom framework is still called for

- **Best Practice 1:** unless you have unusual requirements, use an open source framework

Open Source Framework Features

- Cross browser support
- JavaScript packaging and compression
- Localization
- Accessibility
- Pre-built widgets
- Ajax utilities
- …

Using an Open Source Framework

- Best Practices
  - Public vs. Private
  - Namespacing
  - Versioning
  - Support
  - Licensing
- Case Studies:
  - WebLogic Portal and Dojo
  - APEX and jQuery

Public vs. Private

- Can developers use the open source framework you have embedded?
  - Just because it is discoverable, does not make it public
  - Litmus test: will you document code samples showing use of the framework?
- Will see why making it public is a big issue…

- **Best Practice 2:** prefer to keep the embedded framework as a private implementation detail.

Namespacing

- Will you repackage the framework into your own namespace?
- Pros
  - Allows developer to use a different version of the framework
  - Reduces need for you to deliver the “right” version of the framework
  - If private, namespacing reinforces the idea that it is an internal implementation detail

Namespacing

- Cons
  - Hinders the ability for developers to easily incorporate bug fixes from the project
  - May cause extra network traffic if developer uses non-namespaced version of same framework
  - May unwind CDN optimizations

- **Best Practice 3:** prefer to namespace the framework, as it gives developers more flexibility to use whatever they want

Versioning

- What version of the framework do you select?
  - Multiple considerations…
- What is your backwards compatibility story?
  - If a developer writes code against the public framework, what happens with your next release?
  - Not all frameworks promise backwards compatibility
  - You may be locked into the same version of the framework, forever

Versioning

- Will you have to fix the bugs in the open source framework?
  - Do your customers expect this from you?
  - You have to have a story:
    - You provide the fixes as a patch to your product, or
    - Allow the customer to uplevel the framework version on their own
  - What if the bug is in a part of the framework your code doesn't use?
    - If its public, are you obligated to fix it?
  - If using an obscure framework, more likely you have to roll your own fixes

Versioning

- Do you allow customers to uplevel the embedded version?
  - Pro: allows them to adopt bug fixes, enhancements without work on your part
  - Con: if your product code is using the framework, it may break your code
- **Best Practice 4:** whatever you decide, make sure:
  - All stakeholders are involved in the decision
  - Your support policies are clearly established

Licensing

- Different open source frameworks are offered under different licenses

- **Best Practice 5:** make sure the framework you choose has a license that is acceptable to your product

Case Study: WLP 10.2 and Dojo (DnD)

- Portal Framework Drag and Drop (DnD)
  - Provides the ability to drag and drop portlets on a page
- Narrow subset of Dojo .43
- Namespaced and private
- Supported, though no fixes have been required yet
- Upleveling not supported

Case Study: WLP 10.2 and Dojo (DVT)

- Dynamic Visitor Tools (DVT)
  - Provides the portlet picker, inline page editing, etc
- Full Dojo 1.0
  - Serious debate between .43 and 1.0
- Entire feature offered as a “Sample” for first release to side step the complex issues
- All source code provided
- Dojo is public and not namespaced
- Customers may uplevel Dojo as they wish
- No official support for DVT or Dojo framework

Case Study: WLP 2009 and Dojo (DVT)

- Next version of WLP will productize the DVT
- Dojo 1.1
- Semi-private and namespaced ("multiversioned")
  - Limited customizations to standard DVT are allowed to use the namespaced version
    - But may not have forward compatibility with newer WLP releases
  - Any major work must use a customer provided toolkit
- Customers will be allowed to uplevel the framework to any Dojo 1.x beyond 1.1
  - Upleveling will be the Dojo bug fix story
  - WLP will fix issues in it's code that arise from this
  - Relies on Dojo's backwards compatibility story

Case Study: APEX 4.0 and jQuery

- APEX will embed jQuery for both internal use and customer use cases
- jQuery 1.3 (or later, depends on 4.0 release date)
- Public
- Not namespaced
- Upleveling will not be officially supported, though probably will just work
  - jQuery bugs will be handled on a case by case basis

Building a Custom Framework

- When does this make sense
- Case Studies
  - ADF Faces Rich Client
  - AquaLogic Open Controls
  - APEX Ajax Framework
  - WebLogic Portal Declarative Ajax

Building a Custom Framework

- Opposes the first best practice (use an open source one) but sometimes makes sense
  - You might have unique requirements
  - Once built, consider providing to open source

- **Best Practice 6:** when choosing to build a custom framework, make sure you have identified the reasons for doing so, and have exhausted all open source candidates

Building a Custom Framework

- Some of the open source issues apply:
  - Public vs. Private – still a decision to be made
- But most don't:
  - Namespacing – by definition, it is namespaced
  - Support – of course you will be providing the fixes
  - Backwards compatibility – in your hands
  - Versioning – customers cannot uplevel on own
- A few new ones:
  - Cross browser support, localization, etc.
  - Documentation and sample code

Case Study: ADF Faces Rich Client

- Custom built JSF components
  - Did not use Tomahawk, Ajax4JSF etc
  - Did not use open source Ajax/JS framework
- Timing and ownership were major requirements
  - Foundation for Fusion Apps and Middleware
  - Need to own our own destiny
  - Built custom from the ground up
- Contributed back to open source (Trinidad)

Case Study: AquaLogic Open Ctrls

- Custom built JSF components
  - Did not use Tomahawk, Ajax4JSF etc
  - Did not use open source Ajax/JS framework
- Timing and ownership were major requirements
  - Started in 2005
  - Internally used by new BEA products
- Unique in that they are cross compiled as .NET Web Parts for .NET based products

Case Study: APEX Ajax Framework

- APEX 2.0 introduced SQL Workshop
  - Ajax powered interactive SQL editor
  - Built small custom framework to support unique requirements invoking HTTP requests at mod_plsql
- On public forums, customers were thrilled by Workshop, wanted to use same features
- APEX team decided to open up framework to the public, now heavily used by customers
- Will cover in more detail in a later section…

Case Study: WLP Declarative Ajax

- Feature enables WLP customers to just say “Ajax = yes” on a portlet/page, and framework handles the rest
- Ajax framework is small and very targeted:
  - Finds portal URLs in portlet markup
  - Rewrites URLs to use XHR to WLP’s Ajax servlet
  - Handles updating the DOM with the response
- Officially private in 9.2
  - But enables a very powerful use case so became public in a later release
- Will cover in more detail in a later section…

Agenda

- Introducing Oracle Case Studies
- Framework Selection Best Practices
  - Open Source vs. Custom
- **Framework Features Best Practices**
  - **Client side JavaScript API**
  - **Client side eventing**
  - **Enhancing the XmlHttpRequest**
- Related Best Practices
  - RESTful data APIs
  - Single Origin Policy

Client Side JavaScript API

- Complements the core framework (open source or custom)
  - Sits on top of the core Ajax/JavaScript framework
  - Aids developers in quickly building features
- Provides domain specific capabilities, like:
  - Application data model
  - Application actions
  - Application lifecycle hooks

Client Side JavaScript API

- Similar issues as with building a custom Ajax/JavaScript framework
  - Support
  - Documentation
  - Backwards compatibility

- **Best Practice 7:** provide a public domain-specific client side API in addition to the core framework

Case Study: WLP DISC

- DISC is the public client side API for WLP
- Provides
  - Access to the portal page's data model
    - What portlet ID am I? What page ID am I on?
    - Often feeds data to invocations of WLP REST APIs
  - Actions such as portlet refresh (next release)
  - Hooks such as page rendering lifecycle

Case Study: APEX 4.0 Page Model

- APEX 4.0 will include a lightweight JavaScript API
- Will provide data model of the current page
  - What regions are visible, hidden
- Intends to support some light interactions, not persistent page customization

Client Side Eventing

- Enables wireup and communication between elements on the page
- Very useful for developers to inject new components and JavaScript onto a page that interact with existing elements
- Examples:
  - OpenAjax Hub publish/subscribe
  - Dojo publish/subscribe, Dojo connect

Client Side Eventing

- Drawbacks
  - No standard way to do this yet
  - Need to solve data marshalling issues

- **Best Practice 8:** provide a mechanism for your components to publish and subscribe to page events

Case Study: ADF Faces Rich Client

- Enables developers to add event hooks into the ADF components
- Hooks execute arbitrary JavaScript provided by the developer
- Enables developer to override default behavior of the components

Case Study: ADF Faces Rich Client

JSF Document:

```
<af:menu text="Main Menu 1">
    <af:commandMenuItem text="Sub Menu 1_1"/>
    <af:clientListener method="show" type="mouseOver"/>
</af:menu>
```

Custom JavaScript

```
function show(event){
    var adfRichMenu = event.getSource();
    adfRichMenu.getPeer().show();
}
```

Enhancing the XmlHttpRequest

- XmlHttpRequest is the foundation of Ajax
- At times, this is the best place to provide functionality
- May require integration with your Ajax framework of choice

- **Best Practice 9:** consider enhancing the XmlHttpRequest for fine grained solutions

Case Study: WLP XmlPortletRequest

- Remember the declarative Ajax feature discussed previously
- When defined at the page level, multiple portlets may need to be re-rendered based on single request
- WLP provides enhanced XHR to handle multiple markup payloads in the response

Case Study: WLP XmlPortletRequest

```
<render:jspContentUrl contentUri="/myportlet/data.jsp" var="dataUrl"/>
<script type="text/javascript">
    var dataUrl = '${dataUrl}';
    var xmlhttp = new bea.wlp.disc.io.XMLHttpRequest();

    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            // this portlet's markup is in xmlhttp.responseText
        }
    }

    xmlhttp.open('GET', dataUrl, true);
    xmlhttp.send();
</script>
```

Case Study: APEX apex.ajax

- APEX receives HTTP requests through Apache+mod_plsql
- mod_plsql has very specific requirements on how requests are structured
  - Targeting the correct stored procedure
  - Passing lists of parameters
- Invoking XHR against mod_plsql directly is not easy
- apex.ajax.ondemand() was created as a wrapper that handles the messy details

Case Study: APEX apex.ajax

```
function OnDemand(){
  var apexRequest = new apex.ajax.ondemand('HELLO',
      function(){
          var l_s = p.readyState;
          if(l_s == 1||l_s == 2||l_s == 3){}
          else if(l_s == 4){alert(p.responseText);}
          else{return false;}
      }
  );
  apexRequest._get();
}
```

Agenda

- Introducing Oracle Case Studies
- Framework Selection Best Practices
  - Open Source vs. Custom
- Framework Features Best Practices
  - Client side JavaScript API
  - Client side eventing
  - Enhancing the XmlHttpRequest
- **Related Best Practices**
  - **RESTful data APIs**
  - **Single Origin Policy**

RESTful Data APIs

- Warrants more than a single slide, but here we are
- Think “web friendly API”
- RESTful APIs adhere to the semantics of HTTP to provide power with simplicity
  - URLs map to nouns in your system
  - HTTP verbs are significant: GET, POST, PUT, DELETE
  - Return payload might be XML, JSON, HTML
- In short, and massively oversimplified:
  - RESTful APIs provide data that is easily consumed by a browser via your Ajax framework (XHR)

RESTful Data APIs

- **Best Practice 10:** complement your client side frameworks and API with server side RESTful data APIs.
- Things to pay attention to:
  - Vibrant community discussions about proper REST
    - URL design is time consuming
  - Security concerns
    - Allow APIs to be disabled if customer chooses
    - Guard against CSRF attacks
  - Performance – paging through data sets
  - Providing batch update capabilities

Case Study: WLP and MSFT Popfly

- WLP has an embedded Content Management (CM) system
- Traditionally, not very easy to surface this content in other applications
  - Programmatic integration required
- Added RESTful APIs to WLP CM, and it opens up many possibilities
- Microsoft Popfly is a hosted JavaScript application creation tool
- RESTful APIs enables Popfly (and many other clients) to consume WLP content via XHR

Single Origin Policy

- Limits the reach of the XmlHttpRequest
  - Helps defend against malicious XSS attacks
  - There are ways around it, but...
- Consider proxy solutions if your product requires XHRs to different network domains
  - Custom building a proxy
  - Rely on an existing proxy solution (Squid, etc)
- Think carefully how to lock it down
  - Open proxies are dangerous
  - White lists, black lists
  - Defending against URL hacking
- **Best Practice 11:** don't build your own proxy

Case Study: WLP WSRP Proxy

- Not a proxy that helps with the Single Origin Policy, but an interesting case study
- WSRP is an important portal standard
  - Long story short – requires implementers to build a proxy servlet
- Issues encountered:
  - Securing the proxy servlet
  - Cookie management
  - Opaque vs. transparent URLs

Questions?


ORACLE®