BUILDING AN AJAX INTERFACE WITH ZF

By Wil Sinclair, Zend Framework Development Manager
Matthew Weier O’Phinney, Zend Framework Software Architect
We’ll be covering...

• The nature of the Zend Framework and Dojo Toolkit collaboration
• How- and where- to build integration points in to ZF for any JavaScript toolkit
• How to use these integration points to make building your next AJAX user interface easier
Zend and Dojo

• Partnering to deliver the best out-of-box AJAX experience available in any server-side framework
• Non-exclusive partnership
• While Dojo was the right choice for Zend to partner with, it may not be the right choice for you
  ▪ Different JavaScript toolkits have different strength and weaknesses
  ▪ Both JavaScript toolkits and the requirements of applications that use them are complex; may be best to go with what you know
  ▪ It may not be your choice!
Zend and ?

• Zend Framework has been built with extensibility in mind; this come in handy when integrating with other web application technologies, including JavaScript toolkits

• Zend Framework developers have already written applications using several popular JavaScript toolkits
  ▪ Prototype/Scriptaculous
  ▪ YUI
  ▪ jQuery
  ▪ ?
Integration Points in ZF

- Protocol support (e.g., JSON-RPC)
- View Helper for setting up the client-side environment
- Components for library-specific data envelopes (e.g., dojo.data, YUI DataSource)
- AJAX-enabled Form Elements
Protocol Support

• RPC protocols in JavaScript toolkits make client creation easier, Zend Framework can make server creation easier
• For Zend Framework 1.6 we’ll be implementing a JSON-RPC server
• Other possibilities include JSONP-RPC, Bayeux Protocol, etc. servers
• Not as simple as it sounds
• Different libraries and stylesheets must be included, download location may be need to be specified, themes may need to be specified, etc.
• Much of this is done to optimize page load times
• Many options that are set for the entire page may be set here, such as specifying a theme
Components for Data Envelopes

- Each library typically has its own data envelope.
- Examples include dojo.data and YUI DataSource, but hardly end there.
- ZF can make dealing with these different kinds of data envelopes simpler by providing an interface to package data payloads using these library-specific data envelopes.
- One interface to rule them all?
AJAX-enabled Form Elements

• Common AJAX-enabled form elements include those with auto-completion and immediate validation
• Widgets from the JS toolkits should be easy to use with Zend_Form
• In some cases existing form elements may simply be decorated with AJAX functionality
• New form elements can be created to specifically model behavior made possible by AJAX/DHTML
• There is literally no end to the possibilities here
PROTOCOL SUPPORT: JSON-RPC

JSON-RPC is a lightweight RPC utilizing JSON for its message payloads.
Why JSON-RPC?

- Lightweight protocol
- Seamless mapping of remote methods to local javascript objects
- Easy handling of results and errors
$server = new Zend_Json_Server();

// Specify SMD metadata
$server->setTransport('POST')
    ->setTarget('/unit-test/json-rpc')
    ->setEnvelope(Zend_Json_Server_Smd::ENV_JSONRPC_2);

// Specify class to serve
$server->setClass('My_TestRunner');
Returning a Service Mapping Description

- Service Mapping Description tells JSON-RPC client what methods and arguments are available

```php
$smd = $server->getServiceMap();
$smd->setDojoCompatible(true);
echo $smd->toJson();
```
Handling the request

- JSON-RPC server follows the SoapServer API; handle() will create its request and response by default and emit it – though this behavior is configurable.

```php
(server)->handle();
```
Setting things up is half the battle when integrating Javascript in your applications.
Without view helpers, setting up your javascript includes can be painful:

```php
$this->headScript()
    ->appendFile($this->baseUrl() . '/javascript/dojo/dojo.js')
    ->captureStart(); ?>
dojo.require("dojo.rpc.JsonService");
<?
$this->headScript()->captureEnd();
```
With view helpers

- Creating view helpers makes things easier:

```php
$this->dojo()
    ->useLocalPath('/javascript/dojo/dojo.js')
    ->dojoRequire('dojo.rpc.JsonService');
```
Data Envelopes
A good JavaScript toolkit has data abstraction; supporting this on the server side allows easy integration.

DATA ENVELOPES
What do data envelopes do?

- Abstracts data result sets
- Provides standard accessors to data
- Allows switching data stores on the server seamlessly
Use cases for data envelopes

• Grid components
• Filtering selects (utilizing id/label pairs)
• Trees
Potential data envelope interface

// Where $resultSet is a Zend_Db_Table_Rowset
$data = new Zend_Dojo_Data($resultSet);
$data->setFields($resultSet->getTable()->info(Zend_Db_Table_Abstract::COLUMNS))
    ->setIdentifier('id')
    ->setLabel('title');
echo $data;
AJAX-ENABLED FORM ELEMENTS

Automate creation of AJAX form widgets
How?

- Create targeted Zend_Form_Element implementations for your Javascript toolkit
- Have the decorators call on the view helpers for your Javascript toolkit, thus setting up the Javascript environment
- Benefit from a better UI
Potential elements

- Date/Time selectors
- Autocompleting combo and select boxes
- Currency input
- Range sliders
- Color selectors
- Dialogs and tooltips
- Grids
- Layout elements (tabs, panes, etc) (used for sub forms, display groups, etc.)
$form = new Zend_Form();
$form->addElement('Dojo_DateTextBox', 'date')
    ->addElement('Dojo_TimeBox', 'time')
    ->addElement('Dojo_ColorPalette', 'color')
    ->addElement('Dojo_FilteringSelect', 'choose', array(  
        'dataSource' => '/foo/choose-completion',  
    ));
AJAX-ENABLING
ZF MVC APPLICATIONS

Putting it all together
How?

- Use Zend_Form
- Use Autocomplete action helpers
- Use AjaxContext action helper
- Create actions that handle JSON-RPC requests
Adding JSON-RPC functionality

- All GET requests return SMD descriptions
- All POST requests are handled as JSON-RPC
public function getJsonRpcServer()
{
    // Disable auto-rendering and layouts
    $this->_helper->viewRenderer->setNoRender(true);
    $this->_helper->layout->disableLayout(true);

    $server = new Zend_Json_Server();

    // Specify SMD metadata
    $server->setTransport('POST')
        ->setTarget('/unit-test/json-rpc')
        ->setEnvelope(Zend_Json_Server_Smd::ENV_JSONRPC_2);

    // Specify class to serve
    $server->setClass('My_TestRunner');

    // Disallow auto-emitting response
    $server->setAuto EmitResponse(false);

    return $server;
}
Create a JSON-RPC action

```php
public function jsonRpcAction()
{
    $server = $this->getJsonRpcServer();
    $response = $this->getResponse();
    if ($this->getRequest()->isGet()) {
        $smd = $server->getServiceMap();
        $smd->setDojoCompatible(true);
        $response->setHeader('Content-Type', 'application/json')
            ->setBody($smd->toJson());
    } else {
        $result = $server->handle();
        $response->setBody($result->toJson());
    }
}
```
Setting up an AutoComplete element

```php
->addElements(array(
    'test' => array('type' => 'text', 'options' => array(
        'validators' => array(
            array('validator' => 'Regex', 'options' => array('/^[a-z][a-z_]+$i')),
            'TestName',
        ),
        'filters' => array('StringTrim'),
        'dojoType' => array('dijit.form.ComboBox'),
        'store' => 'testStore',
        'autoComplete' => 'false',
        'hasDownArrow' => 'true',
        'label' => 'Please select a unit test and press the 'Run' button:',
    )),
```
public function autocompleteAction()
{
    if ('ajax' != $this->_getParam('format', false)) {
        return $this->_helper->redirector('index');
    }
    if ($this->getRequest()->isPost()) {
        return $this->_helper->redirector('index');
    }

    require_once 'My/TestFinder.php';
    $match = trim($this->getRequest()->getQuery('test', ''));
    if ('*' == substr($match, -1)) {
        $match = rtrim($match, '*');
    }
    $finder = new My_TestFinder();

    if (empty($match)) {
        $matches = $finder->getTests();
    } else {
        $matches = $finder->getMatchingTests($match);
    }
    return $this->_helper->autoCompleteDojo($matches);
}
Let’s look at an example...