Zend_Queue - Justin Plock

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Queue Component Proposal

Proposed Component Name	Zend_Queue
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Queue
Proposers	Justin Plock Daniel Lo
Revision	1.0 - 3 March 2008: Initial revision 1.1 - 27 February 2009: Updated proposal 1.2 - 9 April 2009: Code complete (wiki revision: 15)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Queue is a standardized interface for dealing with a variety of queuing systems. Proposed systems include: simple array access, Zend_Cache, Zend Platform Job Queue, Amazon's Simple Queue Service (SQS). It should support creating queues, determining the number of messages in a queue, retrieving messages from a queue (all or specific number), submitting messages to a queue, and removing queues.

2. References

- Amazon Simple Queue Service (SQS)
- Amazon SQS Getting Started Guide
- Amazon SQS Developer Guide
- Visibility Timeout
- Zend Platform Job Queue
- FIFO

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **should** support a variety of backend services (Zend_Db, Zend Platform Job Queue, Arrays, MemcacheQ, etc).
- This component **will** support the full Amazon SQS API
- This component **will** support creating new queues
- This component **will** support deleting existing queues
- This component **will** support counting the number of a messages in a queue
- This component **will** support retrieving messages from a queue (all or a specific count)
- This component **will** support submitting messages to a queue

4. Dependencies on Other Framework Components

- Zend_Http_Client
- Zend_Log
- Zend_Db

5. Theory of Operation

Zend_Queue will be a factory class for to create a specific queue interface class. Simple "send" and "receive" methods will be employed by each backend to interact with queues. "create", "getCount" and "delete" methods will be used to manage queues.

6. Milestones / Tasks

- Milestone 1: [DONE] Design notes will be published here
- Milestone 2: [DONE] Working prototype checked into the incubator supporting use cases
- Milestone 3: [DONE] Unit tests exist, work, and are checked into SVN.
- Milestone 4: [DONE] Initial documentation exists.

7. Class Index

- Zend_Queue_Exception
- Zend_Queue (factory class)
- Zend_Queue_Message
- Zend_Queue_Message_Iterator
- Zend_Queue_Adapter_Abstract
- Zend_Queue_Adapter_Exception
- Zend_Queue_Adapter_Amazon (interface for Amazon SQS)
- Zend_Queue_Adapter_Db (interface for Zend_Db)
- Zend_Queue_Adapter_Memcacheq (interface for MemcacheQ)
- Zend_Queue_Adapter_Null
- Zend_Queue_Adapter_Stomp
- Zend_Queue_Adapter_Stomp_IO
- Zend_Queue_Adapter_Array

8. Use Cases

UC-01

```
$config = array(
'name'           => 'queue1',
'driver_options' => array(
'access_key' => AWS_ACCESS_KEY,
'secret_key' => AWS_SECRET_KEY
)
);
$queue = Zend_Queue::factory('Amazon', $config);

// Get list of queues
foreach ($queue->getQueues() as $name) {
echo "$name\n";
}

// Create a new queue
$queue->create('queue2');

// Get number of messages in a queue (supports Countable from SPL)
echo count($queue);

// Get up to 5 messages from a queue
$messages = $queue->receive(5);

foreach ($messages as $message) {
echo $message->body."\n";

// When a message is retrieved from the queue, it is not deleted,
// but will be hidden from subsequent receive() requests for a "visibility timeout"
// (see "Visibility Timeout" above). Once the visibility timeout has expired, another
// receive() request will retrieve the message again. When a message is returned, a
// combination of the requester plus the messageId are returned as a "message handler".
// This handler is a unique instance of a specific messageId being returned to a specific
// requester at a specific date/time. Once processing of the message has been completed,
// the requester can notify the queue to delete the message.
$queue->deleteMessage($message->handle);
}

// Send a message to the currently active queue
$queue->send('My Test Message');

// Delete a queue and all of it's messages
$queue->delete('queue2');
```

9. Class Skeletons

```
class Zend_Queue_Exception extends Zend_Exception {}

abstract class Zend_Queue {

/**
* Use the TIMEOUT constant in the config of a Zend_Queue_Adapter.
*/
const TIMEOUT = 'timeout';

/**
* Factory for Zend_Queue_Adapter_Abstract classes.
*
* First argument may be a string containing the base of the adapter class
* name, e.g. 'Amazon' corresponds to class Zend_Queue_Adapter_Amazon.  This
* is case-insensitive.
```

```
*
* First argument may alternatively be an object of type Zend_Config.
* The adapter class base name is read from the 'adapter' property.
* The adapter config parameters are read from the 'params' property.
*
* Second argument is optional and may be an associative array of key-value
* pairs.  This is used as the argument to the adapter constructor.
*
* If the first argument is of type Zend_Config, it is assumed to contain
* all parameters, and the second argument is ignored.
*
* @param  mixed $adapter String name of base adapter class, or Zend_Config object.
* @param  mixed $config  OPTIONAL; an array or Zend_Config object with adapter parameters.
* @return Zend_Queue_Adapter_Abstract
* @throws Zend_Queue_Exception
*/
public static function factory($adapter, $config = array())
{
/*
* Convert Zend_Config argument to plain string
* adapter name and separate config object.
*/
if ($adapter instanceof Zend_Config) {
if (isset($adapter->params)) {
$config = $adapter->params->toArray();
}
if (isset($adapter->adapter)) {
$adapter = (string)$adapter->adapter;
}
else {
$adapter = null;
}
}

/*
* Verify that adapter parameters are in an array.
*/
if (!is_array($config)) {
/**
* @see Zend_Queue_Exception
*/
require_once 'Zend/Queue/Exception.php';
throw new Zend_Queue_Exception('Adapter parameters must be in an array or a Zend_Config object');
}

/*
* Verify that an adapter name has been specified.
*/
if (!is_string($adapter) || empty($adapter)) {
/**
* @see Zend_Queue_Exception
*/
require_once 'Zend/Queue/Exception.php';
throw new Zend_Queue_Exception('Adapter name must be specified in a string');
}

/*
* Form full adapter class name
*/
$adapterNamespace = 'Zend_Queue_Adapter';
if (isset($config['adapterNamespace'])) {
$adapterNamespace = $config['adapterNamespace'];
unset($config['adapterNamespace']);
}
$adapterName = strtolower($adapterNamespace . '_' . $adapter);
$adapterName = str_replace(' ', '_', ucwords(str_replace('_', ' ', $adapterName)));

/**
* @see Zend_Loader
*/
```

```
require_once 'Zend/Loader.php';
Zend_Loader::loadClass($adapterName);

/*
* Create an instance of the adapter class.
* Pass the config to the adapter class constructor.
*/
$queueAdapter = new $adapterName($config);

/*
* Verify that the object created is a descendent of the abstract adapter type.
*/
if (! $queueAdapter instanceof Zend_Queue_Adapter_Abstract) {
/**
* @see Zend_Queue_Exception
*/
require_once 'Zend/Queue/Exception.php';
throw new Zend_Queue_Exception("Adapter class '$adapterName' does not extend
Zend_Queue_Adapter_Abstract");
}

return $queueAdapter;
}

}

abstract class Zend_Queue_Adapter_Abstract
{

/**
* User-provided configuration
*
* @var array
*/
protected $_config = array();

/**
* Visibility Timeout (seconds)
*
* @var integer
*/
protected $_timeout = 30;

/**
* Zend_Queue_Message class
*
* @var string
*/
protected $_msgClass = 'Zend_Queue_Message';

/**
* Zend_Queue_Message_Iterator class
*
* @var string
*/
protected $_msgsetClass = 'Zend_Queue_Message_Iterator';

/**
* Constructor.
*
* $config is an array of key/value pairs or an instance of Zend_Config
* containing configuration options.  These options are common to most adapters:
*
* name           => (string) The name of the queue to use
* timeout        => (int) Visibility timeout to use (default 30 seconds)
*
* Some options are used on a case-by-case basis by adapters:
*
* access_key     => (string) Amazon AWS Access Key
* secret_key     => (string) Amazon AWS Secret Key
```

```
* dbname         => (string) The name of the database to user
* username       => (string) Connect to the database as this username.
* password       => (string) Password associated with the username.
* host           => (string) What host to connect to, defaults to localhost
* port           => (string) The port of the database
* persistent     => (boolean) Whether to use a persistent connection or not, defaults to false
* protocol       => (string) The network protocol, defaults to TCPIP
* caseFolding    => (int) style of case-alteration used for identifiers
*
* @param  array|Zend_Config $config An array or instance of Zend_Config having configuration data
* @throws Zend_Queue_Adapter_Exception
*/
public function __construct($config)
{
/*
* Verify that adapter parameters are in an array.
*/
if (!is_array($config)) {
/*
* Convert Zend_Config argument to a plain array.
*/

/**
* @see Zend_Config
*/
require_once 'Zend/Config.php';
if ($config instanceof Zend_Config) {
$config = $config->toArray();
}
else {
/**
* @see Zend_Queue_Exception
*/
require_once 'Zend/Queue/Exception.php';
throw new Zend_Queue_Exception('Adapter parameters must be in an array or a Zend_Config object');
}
}

// we need at least a queue name
if (!array_key_exists('name', $config)) {
/**
* @see Zend_Queue_Adapter_Exception
*/
require_once 'Zend/Queue/Adapter/Exception.php';
throw new Zend_Queue_Adapter_Exception("Configuration array must have a key for 'name' that names the
queue");
}

$options = array(
Zend_Queue::TIMEOUT => $this->_timeout
);
$driverOptions = array();

// normalize the config and merge it with the defaults
if (array_key_exists('options', $config)) {
// can't use array_merge() because keys might be integers
foreach ((array)$config['options'] as $key => $value) {
$options[$key] = $value;
}
}
if (array_key_exists('driver_options', $config)) {
// can't use array_merge() because keys might be integers
foreach ((array)$config['driver_options'] as $key => $value) {
$driverOptions[$key] = $value;
}
}
$this->_config  = array_merge($this->_config, $config);
$this->_config['options'] = $options;
$this->_config['driver_options'] = $driverOptions;
```

```
// obtain timeout property if there is one
if (array_key_exists(Zend_Queue::TIMEOUT, $options)) {
$this->_timeout = (int)$options[Zend_Queue::TIMEOUT];
}
}

/**
* Returns the configuration variables in this adapter.
*
* @return array
*/
public function getConfig()
{
return $this->_config;
}

/**
* Returns the active queue name
*
* @return string
*/
public function getActiveQueue()
{
return $this->_config['name'];
}

/**
* Sets the current active queue
*
* @param  string $name queue name
* @return Zend_Queue_Adapter_Abstract Provides Fluent interface
*/
public function setActiveQueue($name)
{
$this->_config['name'] = $name;
return $this;
}

/**
* Create a new queue
*
* @param  string  $name    queue name
* @param  integer $timeout default visibility timeout
* @return boolean
*/
abstract public function create($name, $timeout=null);

/**
* Delete a queue and all of it's messages
*
* @param  string $name queue name
* @return boolean
*/
abstract public function delete($name);

/**
* Send a message to the queue
*
* @param  string $message Message to send to the active queue
* @return string
* @throws Zend_Queue_Adapter_Abstract
*/
abstract public function send($message);

/**
* Get messages in the queue
*
* @param  integer $max_msgs Maximum number of messages to return
* @param  integer $timeout  Visibility timeout for these messages
* @return array
```

```
* @throws Zend_Queue_Adapter_Abstract
*/
abstract public function receive($max_msgs=null, $timeout=null);

/**
* Delete a message from the queue
*
* @param  string $handler
* @return boolean
*/
abstract public function deleteMessage($handler);

/**
* Get an array of all available queues
*
* @return array
* @throws Zend_Queue_Adapter_Abstract
*/
abstract public function getQueues();

/**
* Return the approximate number of messages in the queue
*
* @return integer
* @throws Zend_Queue_Adapter_Abstract
*/
public function count()
{
}
}

class Zend_Queue_Message {

/**
* The data for the queue message
*
* @var array
*/
protected $_data = array();

/**
* Connected is true if we have a reference to a live
* Zend_Queue_Adapter_Abstract object.
* This is false after the Message has been deserialized.
*
* @var boolean
*/
protected $_connected = true;

/**
* Zend_Queue_Adapter_Abstract parent class or instance
*
* @var Zend_Queue_Adapter_Abstract
*/
protected $_queue = null;

/**
* Name of the class of the Zend_Queue_Adapter_Abstract object.
*
* @var string
*/
protected $_queueClass = null;

/**
* Constructor
*
* @param array $config
* @throws Zend_Queue_Exception
*/
public function __construct(array $config=array())
```

```
{
if (isset($config['queue']) && $config['queue'] instanceof Zend_Queue_Adapter_Abstract) {
$this->_queue = $config['queue'];
$this->_queueClass = get_class($this->_queue);
}
if (isset($config['data'])) {
if (!is_array($config['data'])) {
/**
* @see Zend_Queue_Exception
*/
require_once 'Zend/Queue/Exception.php';
throw new Zend_Queue_Exception('Data must be an array');
}
$this->_data = $config['data'];
}
}

/**
* Store queue and data in serialized object
*
* @return array
*/
public function __sleep()
{
return array('_queueClass', '_data');
}

/**
* Setup to do on wakeup.
* A de-serialized Message should not be assumed to have access to a live
* queue connection, so set _connected = false.
*
* @return void
*/
public function __wakeup()
{
$this->_connected = false;
}

/**
* Retrieve message field value
*
* @param  string $key The user-specified key name.
* @return string      The corresponding key value.
* @throws Zend_Queue_Exception if the $key is not a column in the message.
*/
public function __get($key)
{
if (!array_key_exists($key, $this->_data)) {
/**
* @see Zend_Queue_Exception
*/
require_once 'Zend/Queue/Exception.php';
throw new Zend_Queue_Exception("Specified field \"$key\" is not in the message");
}
return $this->_data[$key];
}

/**
* Set message field value
*
* @param  string $key   The message key.
* @param  mixed  $value The value for the property.
* @return void
* @throws Zend_Queue_Exception
*/
public function __set($key, $value)
{
if (!array_key_exists($key, $this->_data)) {
/**
```

```
* @see Zend_Queue_Exception
*/
require_once 'Zend/Queue/Exception.php';
throw new Zend_Queue_Exception("Specified field \"$key\" is not in the message");
}
$this->_data[$key] = $value;
}

/**
* Test existence of message field
*
* @param  string  $key The column key.
* @return boolean
*/
public function __isset($key)
{
return array_key_exists($key, $this->_data);
}

/**
* Returns the queue object, or null if this is disconnected message
*
* @return Zend_Queue_Adapter_Abstract|null
*/
public function getQueue()
{
return $this->_queue;
}

/**
* Set the queue object, to re-establish a live connection
* to the queue for a Message that has been de-serialized.
*
* @param  Zend_Queue_Adapter_Abstract $queue
* @return boolean
* @throws Zend_Queue_Exception
*/
public function setQueue(Zend_Queue_Adapter_Abstract $queue)
{
if ($queue == null) {
$this->_queue = null;
$this->_connected = false;
return false;
}

$queueClass = get_class($queue);
if (! $queue instanceof $this->_queueClass) {
/**
* @see Zend_Queue_Exception
*/
require_once 'Zend/Queue/Exception.php';
throw new Zend_Queue_Exception("The specified Queue is of class $queueClass, expecting class to be
instance of $this->_queueClass");
}

$this->_queue = $queue;
$this->_queueClass = $queueClass;
$this->_connected = true;
return true;
}

/**
* Query the class name of the Queue object for which this
* Message was created.
*
* @return string
*/
public function getQueueClass()
{
return $this->_queueClass;
```

```
}

/**
* Returns the column/value data as an array.
*
* @return array
*/
public function toArray()
{
return $this->_data;
}

/**
* Sets all data in the row from an array.
*
* @param  array $data
* @return Zend_Queue_Message Provides a fluent interface
*/
public function setFromArray(array $data)
{
foreach ($data as $columnName => $value) {
$this->$columnName = $value;
}
```

```
return $this;
}
}
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>