Zend_Db_Table_Plugin - Simon Mundy, Jack Sleight

<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_Db_Table_Plugin Component Proposal

Proposed Component Name	Zend_Db_Table_Plugin
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Db_Table_Plugin
Proposers	Simon Mundy, Jack Sleight Matthew Weier O'Phinney (Zend Liaison)
Revision	0.2 - 23 February 2008 0.1 - 9 December 2007 (wiki revision: 17)

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
Automatic created/modified dates
Registering plugins within model definition
Tracking data changes with an audit logger
Caching fetch/find operations
9. Class Skeletons

1. Overview

Zend_Db_Table_Plugin provides a plugin system for Zend_Db_Table, in order to easily run common business logic across models. Plugins are registered either with the Plugin Broker (to have them attached to all Zend_Db_Table instances) or set individually from within each model class.

2. References

- Zend_Db
- Zend_Controller Plugins

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** allow fine-grain control over behaviours to be added to Zend_Db_Table, Zend_Db_Table_Rowset and Zend_Db_Table_Row read/write operations.

- This component **will** allow for developer-defined namespaced plugins similar to existing Plugin loader behaviour

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Db_Table

5. Theory of Operation

Plugins are designed to allow business logic to be applied to Zend_Db_Table and associated rows/rowsets in an extensible, flexible manner. They act in a manner similar to MVC plugins, allowing access to Table, Rowset and Row objects both before and after read/write operations. Plugins are processed in FIFO order and it is possible to specify plugin loading order during instantiation.

The first method to add plugins is via the Plugin Broker, whereby each plugin attached is automatically added to each Zend_Db_Table upon instantiation. It is recommended to perform this operation prior to dispatch, ideally within a bootstrap - the Plugin Broker can receive a $config object or array to automate this procedure.

The second method is to add them on a per-table basis, either through configuration options or by the 'addPlugin' method. The '_plugin' property will allow array-based configuration (identical to that of the Plugin Broker) to create plugins and optionally pass options per-plugin.

Plugins can also be serialized with their respective Table/Rowset/Rows.

All 'pre' event methods return a null by default. This allows an operation to continue as expected, however this can be overidden by returning a non-null value. This may be useful for situations where the default behaviour is determined to be undesirable - for example, cancelling a 'delete' operation or returning a cached value instead of a 'live' query.

6. Milestones / Tasks

- Milestone 1: DONE Write proposal
- Milestone 2: IN PROGRESS Gather feedback and revise design as necessary
- Milestone 3: Review by the Zend team
- Milestone 4: Develop full implementation and unit tests
- Milestone 5: Initial documentation

7. Class Index

- Zend_Db_Table_Plugin_Interface
- Zend_Db_Table_Plugin_Broker
- Zend_Db_Table_Plugin_Abstract
- Zend_Db_Table_Plugin_Exception

8. Use Cases

9. Class Skeletons

```
interface Zend_Db_Table_Plugin_Interface
{
    public function __construct(array $options = array());

    public function preInitTable(Zend_Db_Table_Abstract $table);
    public function postInitTable(Zend_Db_Table_Abstract $table);

    public function preInitRowset(Zend_Db_Table_Rowset_Abstract $rowset);
    public function postInitRowset(Zend_Db_Table_Rowset_Abstract $rowset);

    public function preInitRow(Zend_Db_Table_Row_Abstract $row);
    public function postInitRow(Zend_Db_Table_Row_Abstract $row);

    public function preFetchTable(Zend_Db_Table_Abstract $table, Zend_Db_Table_Select $select);
    public function postFetchTable(Zend_Db_Table_Abstract $table, Zend_Db_Table_Select $select,
$result);

    public function preSaveTable(Zend_Db_Table_Abstract $table, array $data);
    public function postSaveTable(Zend_Db_Table_Abstract $table, array $data, $result);

    public function preInsertTable(Zend_Db_Table_Abstract $table, array $data);
    public function postInsertTable(Zend_Db_Table_Abstract $table, array $data, $result);

    public function preUpdateTable(Zend_Db_Table_Abstract $table, array $data, $where);
    public function postUpdateTable(Zend_Db_Table_Abstract $table, array $data, $where, $result);

    public function preDeleteTable(Zend_Db_Table_Abstract $table, $where);
    public function postDeleteTable(Zend_Db_Table_Abstract $table, $where, $result);

    public function preSaveRow(Zend_Db_Table_Row_Abstract $row);
    public function postSaveRow(Zend_Db_Table_Row_Abstract $row, $result);

    public function preInsertRow(Zend_Db_Table_Row_Abstract $row);
    public function postInsertRow(Zend_Db_Table_Row_Abstract $row, $result);

    public function preUpdateRow(Zend_Db_Table_Row_Abstract $row);
    public function postUpdateRow(Zend_Db_Table_Row_Abstract $row, $result);

    public function preDeleteRow(Zend_Db_Table_Row_Abstract $row);
    public function postDeleteRow(Zend_Db_Table_Row_Abstract $row, $result);

    public function getColumn(Zend_Db_Table_Row_Abstract $row, $columnName);
    public function setColumn(Zend_Db_Table_Row_Abstract $row, $columnName, $value);
}

abstract class Zend_Db_Table_Plugin_Abstract implements Zend_Db_Table_Plugin_Interface
{
    protected $_options = array();

    public function __construct($options = array())
    {
        $this->setOptions($options);
    }

    public function setOptions($options = array())
    {
        if ($options instanceof Zend_Config) {
            $options = $options->toArray();
        }

        if (!is_array($options)) {
            throw new Zend_Db_Table_Plugin_Exception('Invalid options');
        }

        $this->_options = $options;
    }
```

```
    public function getOptions()
    {
        return $this->_options;
    }

    // ...implements all defined methods in interface as 'empty' methods
}

class Zend_Db_Table_Plugin_Broker
{
    const GLOBAL        = 'global';
    const PREPEND       = 'prepend';
    const APPEND        = 'append';
    const BEFORE_PLUGIN = 'before';
    const AFTER_PLUGIN  = 'after';

    public function addPrefix($prefix)
    {
    }

    public function addPath($path)
    {
    }

    public function addPrefixPath($prefix, $path)
    {
    }

    public function registerPlugin($plugin, $className = self::GLOBAL, $position = self::APPEND,
$positionKey = null)
    {
    }

    public function unregisterPlugin($plugin)
    {
    }

    public function hasPlugin($className)
    {
    }

    public function getPlugin($className)
    {
    }
}
```

```
class Zend_Db_Table_Plugin_Exception extends Zend_Db_Table_Extension
{
}
```

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