Zend_Event - Alvar Vilu

<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_Event Component Proposal

Proposed Component Name	Zend_Event
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Event
Proposers	Alvar Vilu
Zend Liaison	TBD
Revision	1.0 - 2 December 2009: Initial Draft. (wiki revision: 18)

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_Event is AS3'like powerful event system.

2. References

- Github

3. Component Requirements, Constraints, and Acceptance Criteria

- Any event dispatchable component must extend Zend_Event_Dispatcher or implement Zend_Event_Dispatcher_Interface.

4. Dependencies on Other Framework Components

- Zend_Exception

5. Theory of Operation

This component can handle event listeners per obj or via stack trace.

6. Milestones / Tasks

Describe some intermediate state of this component in terms of design notes, additional material added to this page, and / code. Note any significant dependencies here, such as, "Milestone #3 can not be completed until feature Foo has been added to ZF component XYZ." Milestones will be required for acceptance of future proposals. They are not hard, and many times you will only need to think of the first three below.

- Milestone 1: design notes will be published here
- Milestone 2: Working prototype checked into the incubator supporting use cases #1, #2, ...
- Milestone 3: Working prototype checked into the incubator supporting use cases #3 and #4.
- Milestone 4: Unit tests exist, work, and are checked into SVN.
- Milestone 5: Initial documentation exists.

If a milestone is already done, begin the description with "[DONE]", like this:

- Milestone #: [DONE] Unit tests ...

7. Class Index

- Zend_Magic_Exception
- Zend_Magic (factory class)
- Zend_Magic_MindProbe
- Zend_Magic_MindProbe_Intent
- Zend_Magic_Action
- Zend_Magic_CodeGen

8. Use Cases

UC-01

```
require_once 'AVlib/Event/Dispatcher.php';

class example_event extends Zend_Event
{

    const EXAMPLE = 'example_example';

    public $example_var;

    public function __construct ( $type, $bubbles = false, $cancelable = false, $example_var ) {

        //construct the parent
        parent::__construct ( $type, $bubbles, $cancelable );
```

```
        //store Your own custom variables
        $this->example_var = $example_var;

    }

    public function cloneEvent ( $eventPhase, $target, $currentTarget ) {
        $evt = parent::cloneEvent ( $eventPhase, $target, $currentTarget );

        /*
         * do something
         * $this->example_var was copied to clone by parent::cloneEvent
         *
         * so you should extend this function only if you got some other cloning to do
         */

        return $evt;
    }
}

class disp1 extends Zend_Event_Dispatcher
{

    function __construct ( ) {

        //we set up a event listener for event 'loaded' and handle them in listener function
        $this->addEventListener ( example_event::EXAMPLE, array (
                $this,
                'listener'
        ) );

        //now we make 10 disp2 objects
        for ( $i = 0; $i < 10; $i++ ) {
            new disp2 ( );
        }
    }

    public function listener ( Zend_Event $event ) {

        echo $event;

    /*
         * if that event is stoppable, i could stop it on this branch or stop
         * it immediately, so it will not bubble further up to (main)
         */

    }
}

class disp2 extends Zend_Event_Dispatcher
{

    public function __construct ( ) {
        $evt = new example_event ( example_event::EXAMPLE, true, false, 'will be catched every time'
);
        $this->dispatchEvent ( $evt );
    }
```

```
}

$d = new disp1 ( );
```

UC-02

```
require_once 'AVlib/Event/Dispatcher.php';

class evt_custom_dispatcher implements Zend_Event_Dispatcher_Interface
{

    /**
     * @var Zend_Event_Dispatcher
     */
    protected $dispatcher;

    public function __construct ( ) {

        $this->dispatcher = new Zend_Event_Dispatcher ( $this );

    }

    public function addEventListener ( $type, $listener, $useCapture = false, $priority = 0,
$useWeakReference = false ) {

        return $this->dispatcher->addEventListener ( $type, $listener, $useCapture, $priority,
$useWeakReference );
    }

    public function removeEventListener ( $type, $listener, $useCapture = false ) {

        return $this->dispatcher->removeEventListener ( $type, $listener, $useCapture );
    }

    public function removeEventListeners ( $type = null ) {

        return $this->dispatcher->removeEventListeners ( $type );
    }

    public function dispatchEvent ( Zend_Event $event ) {

        return $this->dispatcher->dispatchEvent ( $event );
    }

    public function hasEventListener ( $type ) {

        return $this->dispatcher->hasEventListener ( $type );
    }

    public function willEventTrigger ( $type ) {

        return $this->dispatcher->willEventTrigger ( $type );
    }

}
```

9. Class Skeletons

Zend_Event

```
/**
 * @see Zend_Event_Phase
 */
require_once 'AVlib/Event/Phase.php';

/**
 * @author Alvar Vilu <alvar.vilu@msn.com>
 *
 * @property-read obj  $target   Event target object
 * @property-read string $type   Event type
 * @property-read bool  $bubbles  Will event bubble?
 * @property-read obj  $currentTarget Event receiver object
 * @property-read bool  $cancelable  Can stop event propagation?
 * @property-read int  $eventPhase  Current event phase
 * @property-read bool  $stopPropagation Stops event propagation on another branch
 * @property-read bool  $stopImmediatePropagation Stops propagation immediately
 */
class Zend_Event
{

    /**
     * Event: close
     *
     * @var string
     */
    const CLOSE = 'close';

    /**
     * Event: open
     *
     * @var string
     */
    const OPEN = 'open';

    /**
     * Event: loaded
     *
     * @var string
     */
    const LOADED = 'loaded';

    /**
     * Will event bubbles back to (main)
     *
     * @var bool
     */
    protected $_bubbles = false;

    /**
     * Can something stop event propagation
     *
     * @var bool
     */
    protected $_cancelable = false;

    /**
     * Target where the event dispatcher currently dispatch
     *
     * @var obj
     */
    protected $_currentTarget;

    /**
     * Event current phase
     *
     * @see Zend_Event_Phase
     * @var int
     */
    protected $_eventPhase = 0;
```

```
    /**
     * If cancelable and this set true, progagation will immediately stop
     *
     * @var bool
     */
    protected $_stopImmediatePropagation = false;

    /**
     * If cancelable and this set true, propagation will stop after that obj level
     *
     * @var bool
     */
    protected $_stopPropagation = false;

    /**
     * Object that dispatches the event
     *
     * @var obj
     */
    protected $_target;

    /**
     * A constant string that indicates the type of event
     *
     * @var string
     */
    protected $_type;

    /**
     * @param string  $type
     * @param bool  $bubbles
     * @param bool  $cancelable
     * @return Zend_Event
     */
    public function __construct ( $type, $bubbles = false, $cancelable = false ) {

        $this->_type = ( string ) $type;
        $this->_bubbles = ( bool ) $bubbles;
        $this->_cancelable = ( bool ) $cancelable;

    }

    /**
     * Get read-only parameters
     *
     * @param string $name
     * @throws Zend_Event_Exception If event parameter not readable.
     * @return mixed
     */
    final public function __get ( $name ) {

        switch ( $name ) {
            case 'bubbles':
                return $this->_bubbles;
            case 'cancelable':
                return $this->_cancelable;
            case 'currentTarget':
                return $this->_currentTarget;
            case 'eventPhase':
                return $this->_eventPhase;
            case 'stopImmediatePropagation':
                return $this->_stopImmediatePropagation;
            case 'stopPropagation':
                return $this->_stopPropagation;
            case 'target':
                return $this->_target;
            case 'type':
                return $this->_type;
```

```
        default:
            require_once 'AVlib/Event/Exception.php';
            throw new Zend_Event_Exception ( 'Event parameter not readable: ' . $name );
    }

}

/**
 * Set writeable parameters
 *
 * @param string $name
 * @param mixed $value
 * @throws Zend_Event_Exception If event parameter not writable.
 * @return mixed
 */
public function __set ( $name, $value ) {
    require_once 'AVlib/Event/Exception.php';
    throw new Zend_Event_Exception ( 'Event parameter not writable: ' . $name );
}

/**
 * Format event to a pretty string.
 *
 * @return string
 */
public function toString ( ) {

    return 'Zend_Event, type: ' . $this->_type;
}

final public function __toString ( ) {
    return $this->toString ( );
}

/**
 * @param int $eventPhase
 * @param obj $target
 * @param obj $currentTarget
 * @return Zend_Event
 */
public function cloneEvent ( $eventPhase, $target, $currentTarget ) {

    $evt = clone $this;
    $evt->_eventPhase = $eventPhase;
    $evt->_target = $target;
    $evt->_currentTarget = $currentTarget;

    return $evt;
}

/**
 * Clone should only be available inside of event
 */
final protected function __clone ( ) {}

/**
 * Stops event propagation on another branch
 *
 * @return Zend_Event
 */
final public function stopPropagation ( ) {

    if ( $this->_cancelable )
        $this->_stopPropagation = true;

    return $this;
}

/**
 * Stops event propagation immediately
```

```
     *
     * @return Zend_Event
     */
    final public function stopImmediatePropagation ( ) {

        if ( $this->_cancelable )
            $this->_stopImmediatePropagation = true;

        return $this;
```

```
    }

}
```

Zend_Exception

```
/**
 * @see Zend_Exception
 */
require_once 'AVlib/Exception.php';

/**
 * @author Alvar Vilu <alvar.vilu@msn.com>
 *
 */
class Zend_Event_Exception extends Zend_Exception
{}
```

Zend_Event_Dispatcher

```


/**
 * @see Zend_Event_Dispatcher_Interface
 */
require_once 'AVlib/Event/Dispatcher/Interface.php';

/**
 * @see Zend_Event_Phase
 */
require_once 'AVlib/Event/Phase.php';

/**
 * @see Zend_Event
 */
require_once 'AVlib/Event.php';

/**
 * @author Alvar Vilu <alvar.vilu@msn.com>
 *
 */
class Zend_Event_Dispatcher implements Zend_Event_Dispatcher_Interface
{

    /**
     * An array that holds all listeners that have been created.
     *
     * [eventType][objID][useCapture] = array(listener, listener, ...)
     * listener = array (0 => priority, 1 => callback func
     *
     * @var array
     */
    private static $_listeners = array ();

    /**
     * A object that's used to dispatch events,
     * can be $this on obj that interface Zend_Event_Dispatcher_Interface.
     *
     * @var Zend_Event_Dispatcher
     */
    private $dispatchObj = null;

    public function __construct ( $dispatchObj = null ) {

        $this->dispatchObj = $dispatchObj;
```

```
    }

    final private function _getDispatchObj ( ) {

        if ( null === $this->dispatchObj )
            $this->dispatchObj = $this;

        return $this->dispatchObj;
    }

    /**
     * Add new event listener to spesific type.
     * If use capture is enabled then event will flow from (main) to target
     * The higher the priority number - event will sorted to first.
     * If two or more same priority level events added then first one will be dispatched.
     *
     * @param string $type
     * @param string|array $listener
     * @param bool $useCapture
     * @param int $priority
     * @param bool $useWeakReference UNIMPLEMENTED!
     * @return Zend_Event_Dispatcher Event dispatchable object
     */
    final public function addEventListener ( $type, $listener, $useCapture = false, $priority = 0,
$useWeakReference = false ) {

        $objHash = spl_object_hash ( $this->_getDispatchObj ( ) );
        $useCapture = ( bool ) $useCapture;
        $priority = ( int ) $priority;
        $useWeakReference = ( bool ) $useWeakReference;

        //Prepare self::$_listeners array indexes.
        if ( ! isset ( self::$_listeners[ $type ][ $objHash ][ $useCapture ] ) ) {

            if ( ! isset ( self::$_listeners[ $type ][ $objHash ] ) ) {

                if ( ! isset ( self::$_listeners[ $type ] ) ) {

                    self::$_listeners[ $type ] = array ();
                }

                self::$_listeners[ $type ][ $objHash ] = array ();
            }

            self::$_listeners[ $type ][ $objHash ][ $useCapture ] = array ();
        }

        //If ve have that same listner, don't add it twice.
        foreach ( self::$_listeners[ $type ][ $objHash ][ $useCapture ] as $l ) {

            if ( $l[ 1 ] === $listener ) {

                return $this;
            }
        }

        //Finally add the listener.
        self::$_listeners[ $type ][ $objHash ][ $useCapture ][ ] = array (
                0 => $priority,
                1 => $listener
        );

        //Now rearrange listeners by priority
        rsort ( self::$_listeners[ $type ][ $objHash ][ $useCapture ] );

        return $this;
    }

    /**
```

```
     * @param string $type
     * @param string|array $listener
     * @param bool $useCapture
     * @return Zend_Event_Dispatcher
     */
    final public function removeEventListener ( $type, $listener, $useCapture = false ) {

        $objHash = spl_object_hash ( $this->_getDispatchObj ( ) );
        $useCapture = ( bool ) $useCapture;

        if ( ! isset ( self::$_listeners[ $type ][ $objHash ][ $useCapture ] ) )
            return $this;

        foreach ( self::$_listeners[ $type ][ $objHash ][ $useCapture ] as $k => $l ) {
            if ( $l[ 1 ] === $listener ) {
                unset ( self::$_listeners[ $type ][ $objHash ][ $useCapture ][ $k ] );
                break;
            }
        }

        return $this;
    }

    /**
     * @param string|array|null $type If null, all listeners will be removed.
     * @return Zend_Event_Dispatcher
     */
    final public function removeEventListeners ( $type = null ) {

        $objHash = spl_object_hash ( $this->_getDispatchObj ( ) );

        if ( null === $type ) {
            $types = array_keys ( self::$_listeners );
        }
        elseif ( is_array ( $type ) ) {
            $types = $type;
        }
        else {
            $types = array (
                    ( string ) $type
            );
        }

        foreach ( $types as $type ) {

            if ( isset ( self::$_listeners[ $type ][ $objHash ] ) ) {
                unset ( self::$_listeners[ $type ][ $objHash ] );

                if ( empty ( self::$_listeners[ $type ] ) )
                    unset ( self::$_listeners[ $type ] );
            }
        }

        return $this;
    }

    /**
     * @param Zend_Event $event Instance of Zend_Event to dispatch.
     * @throws Zend_Event_Dispatcher_Exception If listener function returned with a exception.
     * @return bool True if event dispatched successfully.
     */
    final public function dispatchEvent ( Zend_Event $event ) {

        $last = null;
        $dispatchTree = array ();

        //Put 'at target' phase to dispatchTree.
        $objHash = spl_object_hash ( $this->_getDispatchObj ( ) );
        if ( isset ( self::$_listeners[ $event->type ][ $objHash ][ false ] ) ) {
```

```
            $dispatchTree[ ] = array (
                    $this->_getDispatchObj ( ),
                    Zend_Event_Phase::AT_TARGET,
                    self::$_listeners[ $event->type ][ $objHash ][ false ]
            );
        }

        //Prepare dispatch trees for capture and if needed, bubble phase.
        foreach ( debug_backtrace ( ) as $branch ) {

            if ( ! isset ( $branch[ 'object' ] ) or ! $branch[ 'object' ] instanceof
Zend_Event_Dispatcher_Interface )
                break;

            if ( $branch[ 'object' ] === $this or $branch[ 'object' ] === $this->_getDispatchObj ( ) )
                continue;

            if ( $last !== $branch[ 'object' ] ) {

                $last = $branch[ 'object' ];
                $objHash = spl_object_hash ( $last );

                if ( isset ( self::$_listeners[ $event->type ][ $objHash ][ true ] ) ) {

                    array_unshift ( $dispatchTree, array (
                            $last,
                            Zend_Event_Phase::CAPTURING,
                            self::$_listeners[ $event->type ][ $objHash ][ true ]
                    ) );
                }

                if ( $event->bubbles and isset ( self::$_listeners[ $event->type ][ $objHash ][ false
] ) ) {

                    $dispatchTree[ ] = array (
                            $last,
                            Zend_Event_Phase::BUBBLING,
                            self::$_listeners[ $event->type ][ $objHash ][ false ]
                    );
                }

            }

        }

        //Start dispatching.
        $stopPropagation = false;

        try {

            foreach ( $dispatchTree as $dispatcher ) {
                //$dispatcher = array (0 => $obj, 1 => phase, 2 => array(listener, ...))


                foreach ( $dispatcher[ 2 ] as $listener ) {

                    $evt = $event->cloneEvent ( $dispatcher[ 1 ], $this->_getDispatchObj ( ),
$dispatcher[ 0 ] );

                    if ( false === call_user_func ( $listener[ 1 ], $evt ) ) {
                        require_once 'AVlib/Event/Dispatcher/Exception.php';
                        throw new Zend_Event_Dispatcher_Exception ( 'Call to event listener failed' );
                    }

                    if ( true === $evt->stopPropagation ) {
                        $stopPropagation = true;
                    }

                    if ( true === $evt->stopImmediatePropagation ) {
                        return false;
```

```
                    }

                }

                if ( true === $stopPropagation ) {
                    return false;
                }

            }

        } catch ( Exception $e ) {
            require_once 'AVlib/Event/Dispatcher/Exception.php';
            throw new Zend_Event_Dispatcher_Exception ( 'Event dispatching failed', 0, $e );
        }

        return true;
    }

    /**
     * @param string $type
     * @return bool
     */
    final public function hasEventListener ( $type ) {

        return isset ( self::$_listeners[ $type ][ spl_object_hash ( $this->_getDispatchObj ( ) ) ] );
    }

    /**
     * @param string $type Event type
     * @return bool
     */
    final public function willEventTrigger ( $type ) {

        //No events at all for this type
        if ( ! isset ( self::$_listeners[ $type ] ) )
            return false;

        foreach ( debug_backtrace ( ) as $branch ) {

            if ( ! isset ( $branch[ 'object' ] ) or ! ( $branch[ 'object' ] instanceof
Zend_Event_Dispatcher_Interface ) )
                return false;

            if ( isset ( self::$_listeners[ $type ][ spl_object_hash ( $branch[ 'object' ] ) ] ) )
                return true;

        }

        return false;
```

```
    }

}
```

Zend_Event_Exception

```
/**
 * @see Zend_Event_Exception
 */
require_once 'AVlib/Event/Exception.php';

/**
 * @author Alvar Vilu <alvar.vilu@msn.com>
 *
 */
class Zend_Event_Dispatcher_Exception extends Zend_Event_Exception
{}
```

Zend_Event_Dispatcher_Interface

```
interface Zend_Event_Dispatcher_Interface
{

    public function addEventListener ( $type, $listener, $useCapture = false, $priority = 0,
$useWeakReference = false );

    public function removeEventListener ( $type, $listener, $useCapture = false );

    public function removeEventListeners ( $type = null );

    public function dispatchEvent ( Zend_Event $event );

    public function hasEventListener ( $type );

    public function willEventTrigger ( $type );

}
```

Zend_Event_Phase

```
/**
 * @author Alvar Vilu <alvar.vilu@msn.com>
 *
 */
class Zend_Event_Phase
{

    /**
     * main -> target
     *
     * @var int
     */
    const CAPTURING = 1;

    /**
     * at target
     *
     * @var int
     */
    const AT_TARGET = 2;

    /**
     * target -> main
     *
     * @var int
     */
    const BUBBLING = 3;

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