Exception codes in exceptions - Matthew Ratzloff & Thomas Weidner

<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: Exception codes in exceptions Component Proposal

Proposed Component Name	Exception codes in exceptions
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Exception codes in exceptions
Proposers	Matthew Ratzloff Thomas Weidner Alexander Veremyev (Zend Liaison)
Revision	2.0 - October 1, 2007: Revised with community feedback; added Thomas to the list of proposers 2.1 - October 2, 2007: Added translation functionalities 2.2 - October 3, 2007: Added use cases 3.0 - July 20, 2008: Reworked as wished by the devteam (wiki revision: 33)

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

Currently, exceptions can only be handled on a per-class basis, or possibly by string comparison against the message.

Instead, we propose that exception codes be used throughout the framework, using a 4-byte hexadecimal format.

This has at least four obvious benefits.

First, it's now possible to distinguish exceptions based on the type of error instead of only by class. This allows users to handle them intelligently.

Second, codes let us do some interesting things with exception handling. Users would now be able to call a method such as:

```
if ($e->stringTooShort()) {
    ...
}
```

thereby making exception special-casing easy.

Third, this gives us the ability to translate error messages using `Zend_Translate` (loaded only when an exception occurs) and using separate translation files (for example, .mo format). Not all developers speak English; those that do generally prefer to see exception messages in their native language.

Finally, users could potentially visit something like http://framework.zend.com/exceptions/0xFFFFFFFF (easily indexable by search engines) to see a more in-depth description of the exception, as well as possible solutions and even user comments. Exceptions that are commonly encountered can be addressed once, on the website, in an easy to access way. `Zend_Exception`'s constructor, which all exceptions extend, can then be easily modified to provide a direct link to this page.

Error codes can be divided in a predetermined way, so that each component has its own hex value and no overlap occurs. The solution that we've agreed upon is that each component is assigned its own unique range for the first half (e.g., `Zend_Acl` = 0x00020000, `Zend_Auth` = 0x00030000, `Zend_Etc` = 0xFFFF0000), for a maximum of 65,536 possible exception ranges. Each component then has 65,536 exceptions it can create and use. For example, `Zend_Controller`'s first type of exception may be 0x00030000, followed by 0x00030001, 0x00030002, and so on.

2. References

None

3. Component Requirements, Constraints, and Acceptance Criteria

None

4. Dependencies on Other Framework Components

Zend_Translate - Thomas Weidner for translated exceptions via `getTranslatedMessage()`

5. Theory of Operation

Initially, this would require nothing except assignment of hex values to components. As time goes on, the translation system could be implemented, followed by the online system which would use the identical translation files from the framework.

Allocation of exception ranges:

```
0x00000000 to 0x0000FFFF - Zend_Exception (Reserved for PHP bugs and general framework exceptions)
0x00010000 to 0x0001FFFF - Zend_Db
0x00020000 to 0x0002FFFF - Zend_Feed
0x00030000 to 0x0003FFFF - Zend_Filter
0x00040000 to 0x0004FFFF - Zend_Http
0x00050000 to 0x0005FFFF - Zend_Json
0x00060000 to 0x0006FFFF - Zend_Log
0x00070000 to 0x0007FFFF - Zend_Mail
0x00080000 to 0x0008FFFF - Zend_Mime
0x00090000 to 0x0009FFFF - Zend_Pdf
0x000A0000 to 0x000AFFFF - Zend_Service
0x000B0000 to 0x000BFFFF - Zend_Uri
0x000C0000 to 0x000CFFFF - Zend_View
0x000D0000 to 0x000DFFFF - Zend_XmlRpc
0x000E0000 to 0x000EFFFF - Zend_Controller
0x000F0000 to 0x000FFFFF - Zend_Search
0x00100000 to 0x0010FFFF - Zend_Cache
0x00110000 to 0x0011FFFF - Zend_Config
0x00120000 to 0x0012FFFF - Zend_Measure
0x00130000 to 0x0013FFFF - Zend_Date
0x00140000 to 0x0014FFFF - Zend_Locale
0x00150000 to 0x0015FFFF - Zend_Acl
0x00160000 to 0x0016FFFF - Zend_Rest
0x00170000 to 0x0017FFFF - Zend_Server
0x00180000 to 0x0018FFFF - Zend_Request
0x00190000 to 0x0019FFFF - Zend_Session
0x001A0000 to 0x001AFFFF - Zend_Console
0x001B0000 to 0x001BFFFF - Zend_Auth
0x001C0000 to 0x001CFFFF - Zend_Gdata
0x001D0000 to 0x001DFFFF - Zend_Registry
0x001E0000 to 0x001EFFFF - Zend_Translate
0x001F0000 to 0x001FFFFF - Zend_Memory
0x00200000 to 0x0020FFFF - Zend_Validate
0x00210000 to 0x0021FFFF - Zend_Debug
0x00220000 to 0x0022FFFF - Zend_Loader
0x00230000 to 0x0023FFFF - Zend_Version
0x00240000 to 0x0024FFFF - Zend_Currency
```

0x0000 is reserved for the `UNKNOWN` exception, which must be declared in each exception class.

6. Milestones / Tasks

7. Class Index

Zend_Exception

8. Use Cases

Use Case 1: (Class) Throwing an exception

```
throw new Zend_Date_Exception('Date cannot be parsed', Zend_Date_Exception::INVALID_INPUT);
```

Use Case 2: (Class) Throwing an exception with parameters

```
throw new Zend_Date_Exception("Date '%1\$s' cannot be parsed at position %2\$s with sign %3\$s",
Zend_Date_Exception::INVALID_INPUT, $input, $position, $defectChar);
```

Use Case 3: (User) Using the standard exception behavior

```
try {
    $date = new Zend_Date('Invalid input');
} catch (Zend_Date_Exception $e) {
    // Error found
    print 'Code: ' . $e->getCode();
    print 'Message: ' . $e->getMessage();
}
```

Use Case 4: (User) Iterating through different exceptions with a switch block

```
try {
    $date = new Zend_Date('Invalid input');
} catch (Zend_Date_Exception $e) {
    switch ($e->getCode()) {
        case Zend_Date_Exception::INVALID_INPUT:
            $input = time();
            break;
        case Zend_Date_Exception::NO_TIMEZONE:
            $date->setTimezone(date_default_timezone_get());
            break;
        case Zend_Date_Exception::NO_LOCALE:
            print 'No locale was found.  Please configure your browser.';
        case $e::UNKNOWN: // Also valid
            print $e->getMessage();
        default:
            print $e->getTranslatedException();
    }
}
```

Use Case 5: (User) Using the exception special-casing methods

```
try {
    $date = new Zend_Date('Invalid input');
} catch (Zend_Date_Exception $e) {
    if ($e->invalidInput()) {
        print $e->getTranslatedMessage();
    } else if ($e->noLocale()) {
        print $translator->_('No locale was found.  Please configure your browser.');
    }
}
```

Use Case 6: (User) Getting a translated exception message

```
try {
    $date = new Zend_Date('Invalid input');
} catch (Zend_Date_Exception $e) {
    print 'Message: ' . $e->getTranslatedMessage(); // the default locale from the user's browser
}
```

Use Case 7: (User) Translating an exception message to a fixed, expected locale

```
try {
    $date = new Zend_Date('Invalid input');
} catch (Zend_Date_Exception $e) {
    print 'Message: ' . $e->getTranslatedMessage('de'); // German exception message
}
```

Use Case 8: (User) Translating an exception message to one of the auto-locales

```
try {
    $date = new Zend_Date('Invalid input');
} catch (Zend_Date_Exception $e) {
    print 'Message: ' . $e->getTranslatedMessage('auto'); // 'browser' and 'environment' are also
supported
}
```

Use Case 9: (User) Setting a standard locale for all exception objects

```
// In the bootstrap
Zend_Exception::setLocale('de');

// In a class
try {
    $date = new Zend_Date('Invalid input');
} catch (Zend_Date_Exception $e) {
    print 'Message: ' . $e->getTranslatedMessage(); // Returns the German translation of the exception
}
```

In all cases, exceptions are handled thoughtfully and in a very readable way.

9. Class Skeletons

The new `Zend_Exception`:

```


class Zend_Exception extends Exception
{
    /** @const integer Exception code range assigned to this component */
    const ASSIGNED_RANGE = 0x00000000;

    /** @const integer Unknown exception */
    const UNKNOWN = 0x00000000;

    /** @const integer The supplied exception code is not an integer */
    const CODE_NOT_INTEGER = 0x00000001;

    /** @const integer The supplied exception code is out of range */
    const CODE_OUT_OF_RANGE = 0x00000002;

    /** @const integer The given locale does not exist */
    const UNKNOWN_LOCALE = 0x00000003;

    /** @var boolean Translate exception messages? */
    protected $_translateMessages = false;

    /** @var string Locale */
    protected $_locale = 'en';

    /** @var array|null Message parameters */
    protected $_messageParams = null;

    /** @var string Path in which to find message (i.e., translation) files */
    protected $_messagePath = 'Zend/Exception';

    /** @var string Message file format */
    protected $_messageFileFormat = 'gettext';

    /** @var string Message file extension */
    protected $_messageFileExt = 'mo';
```

```
    /**
     * Sets whether or not to translate exception messages.
     *
     * @param boolean $flag Translate exception messages?
     */
    public static function translateMessages($flag = true)
    {
        self::$_translateMessages = $flag;
    }

    /**
     * Sets a new standard locale for all exception objects.
     *
     * @param  string|Zend_Locale $locale Locale object or textual representation
     * @return string Locale (can differ from the given one)
     */
    public static function setLocale($locale = null)
    {
        require_once 'Zend/Locale.php';
        if ($locale instanceof Zend_Locale) {
            self::_locale = $locale->toString();
        } else if (!($locale = Zend_Locale($locale, true))) {
            throw new Zend_Exception("Given locale '%1\$s' does not exist", self::UNKNOWN_LOCALE,
$locale);
        } else {
            self::_locale = $locale;
        }
        return self::_locale;
    }

    /**
     * Returns the standard locale for all exception objects.
     *
     * @return string Locale
     */
    public static function getLocale()
    {
        return self::_locale;
    }

    /**
     * Constructor.
     *
     * Additional parameters may be passed in after the code for message translation purposes.
     *
     * @param string $message Exception message
     * @param integer $code 4-byte hexadecimal exception code constant (e.g., Zend_Exception::EXAMPLE)
     * @param string Additional translation parameters ($message, $code[, $arg1, $arg2, $arg3])
     */
    public function __construct($message = null, $code = 0)
    {
        if (!is_int($code)) {
            throw new Zend_Exception('Exception code must be an integer', self::CODE_NOT_INTEGER);
        }
        if (($code & self::ASSIGNED_RANGE) !== self::ASSIGNED_RANGE) {
            throw new Zend_Exception('Exception code is out of range', self::CODE_OUT_OF_RANGE);
        }

        $this->message = $message;
        $this->code    = $code;

        if (func_num_args() > 2) {
            $this->_messageParams = array_splice(func_get_args(), 2);
        }
    }

    /**
     * Returns an exception message.  If TRANSLATE_MESSAGES is true, this will return the
     * value of {@link getTranslatedMessage} instead.
```

```
     *
     * @return string
     */
    public function getMessage()
    {
        if (self::$_translatedMessages) {
            return $this->getTranslatedMessage();
        } else {
            return parent::getMessage();
        }
    }

    /**
     * Returns a translated exception message.
     *
     * @param  string|Zend_Locale $locale Locale
     * @return string
     */
    public function getTranslatedMessage($locale = null)
    {
        $message = parent::getMessage();

        try {
            require_once 'Zend/Translate.php';
            require_once 'Zend/Locale.php';
            if (!($locale = Zend_Locale::isLocale($locale, true))) {
                $locale = 'en';
            }
            if ($locale instanceof Zend_Locale) {
                $locale = $locale->toString();
            }
            $messagePath = $this->_getMessageFilePath($locale);
            $translator  = new Zend_Translate($this->_messageFileFormat, $messagePath, $locale);
            return vprintf($translator->_($message, $this->_messageParams));
        } catch (Exception $e) {
            return vprintf($message, $this->_messageParams);
        }
    }

    /**
     * Returns an exception code given a valid method name.
     *
     * @param  string  $exceptionName Exception name (as a method call)
     * @return integer Exception code
     */
    public function __call($exceptionName, $value = null)
    {
        return ($this->_getConstant($exceptionName) === $this->code);
    }

    /**
     * Returns the path to the message (i.e., translation) file.
     *
     * For example, given a component of "Zend_Example", this would return
     * "Zend/Exception/en/Example.mo".
     *
     * @return string Message file path
     */
    protected function _getMessageFilePath($locale)
    {
        $componentName = implode('', array_splice(explode('_', get_class($this)), 1, 2));
        $filePath      = $this->_messagePath . DIRECTORY_SEPARATOR . $locale . DIRECTORY_SEPARATOR
                       . $componentName . '.' . $this->_messageFileExt;
        return $filePath;
    }

    /**
     * Retrieves a constant given an exception name.
     *
     * @param  string  $exceptionName Exception name
```

```
     * @return integer Exception code, or self::UNKNOWN if not found
     */
    protected function _getConstant($exceptionName)
    {
        $constantName = get_class($this) . '::' . $this->_formatAsConstant($exceptionName);
        if (!defined($constantName)) {
            return self::UNKNOWN;
        }
        return constant($constantName);
    }

    /**
     * Converts exceptionName to EXCEPTION_NAME.
     *
     * @param  string $exceptionName Unformatted exception name
     * @return string Formatted exception name
     */
    protected function _formatAsConstant($exceptionName)
    {
        $parts = preg_split('/([A-Z][a-z0-9]+)/', $exceptionName, -1,
                            PREG_SPLIT_NO_EMPTY | PREG_SPLIT_DELIM_CAPTURE);
```

```
        return strtoupper(implode('_', $parts));
    }
}
```

An example exception:

```
class Zend_My_Exception extends Zend_Exception
{
    /** @const integer Exception code range assigned to this component */
    const ASSIGNED_RANGE = 0x00210000;

    /** @const integer Unknown exception */
    const UNKNOWN = 0x00210000;

    /** @const integer String too long exception */
    const STRING_TOO_LONG  = 0x00210010;

    /** @const integer String too short exception */
    const STRING_TOO_SHORT = 0x00210011;
}
```

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