Error Handling API

Overview

This document explains the error handling capabilities and semantics of this AddOn. Here's an example of how a failure is handled:

```
local result = someFunction( arg1 )
if result[1] == FAILURE then
      errors:postEntry( result )    -- display the result
end
```

The resulting display would look something like this:

```
Input arg nil: Expected non-nil for arg2 at [SnippetTest.lua:9]
STACK TRACE:
Interface\AddOns\Snippets\SnippetTest.lua:10
Interface\AddOns\Snippets\SnippetTest.lua:14
Interface\AddOns\Snippets\SnippetTest.lua:18
Interface\AddOns\Snippets\SnippetTest.lua:21
Interface\AddOns\Snippets\SnippetTest.lua:27
```

And here's how one might check for failed operation, in this case, an input parameter was nil but the code expected a non-nil value[1].

```
local function someFunction( arg1 )
local result = SUCCESSFUL_RESULT
if arg1 == nil then
      result = errors:setErrorResult( L["ARG_NIL", debugstack()
      return result
end
… continue …
end
```

Next we look at the result table

The Result Data Structure

When values are checked for correctness, the results are written into a result table (see below) and returned to the calling function or displayed

[1] The canned error messages are localized, hence the L["ARG_NIL"] serves as an index to a table of localized strings (the localized strings are found in the Localized directory)

by the error message handler (see below). The default layout of the result table is defined in Errors.lua:

```
result = {
      <Failure or Success>, -- integer
      <Error Message>,        -- string
      <Stack Frame >          -- string
}
```

In Errors.lua a default successful result is defined as follows:

```
SUCCESSFUL_RESULT = {STATUS_SUCCESS, nil, nil}
```

So, for example, this might appear in code as:

```
local function someFunction( … )
local result = SUCCESSFUL_RESULT
… continue …
return result
end
```

Failure or Success

The first element of the result table (result[1]) is one of two constants: SUCCESS or FAILURE whose values are 1 and -1 respectively.

Error Message

The AddOn as a number of predefined, localized error messages in the Locales directory (e.g., enUS.lua). Here, for example, are the localized error messages that arise from function arguments:

```
L["ARG_MISSING"]    = "Input arg missing"
L["ARG_NIL"]        = "Input arg nil"
L["ARG_INVALID"]    = "Input arg invalid or out-of-range"
L["ARG_UNKNOWN"]    = "Input arg unknown"
L["ARG_UNEXPECTED"] = "Input arg unexpected"
L["ARG_WRONGTYPE"]  = "Input arg wrong type"
L["ARG_OUTOFRANGE"] = "Input arg out of range"
```

Stack Frame

The stack frame is obtained by calling the Blizzard function, debugstack() as follows:

```
Local result = errors:setErrorResult( L["ARG_NIL"], debugstack() )
Errors:postEntry( result )
```

Error Methods

Result constructor

```
result = errors:setErrorResult( errorMessage, stackFrame )
```

`result`	{ STATUS_FAILURE, errorMessage, stackFrame }
`STATUS_FAILURE`	An integer constant, -1
`errorMessage`	A string from the localized error table (see enUS.lua in Locales directory.
`stackFrame`	A string returned by Blizzard's debugstack()

The where method

The Errors API provides a method for a programmer to print the location of the execution pointer.

(void) errors.where()

This method writes the location (the file and line number) to the default chat frame when the execution reaches and executes the method. For example, suppose we have a file Deep.lua with the following code:

```
-- FILE: Deep.lua
...
13 local function deeper()
14    deepest()
15    errors.where()
16  end
17 local function deep()
18    deeper()
19 end
20 local function start()
21    deep()
22 end
23 start()
24
…
```

Note that at line 15, errors.where() is called. When the start function is invoked, a couple of subroutines are called and the errors.where() method is called at line 15. At that point errors.where() produces the following output to the default chat frame:

```
    [Deep.lua:15]
```

Programmers embed this method in their code to test whether their code's execution path is executed at that point.

Display the Error

Errors are decoded and displayed in the errors:postEntry() method.

```
    (void) errors:postEntry( result )
```

result	{ STATUS_FAILURE, errorMessage, stackFrame }