**JavaScript Error Handling**

**Rule1:** *Assume your code will fail.*

Assume that the variables used in a condition may get nulls and these situations need to be handled. Please check the below piece of code.

Object.extend = function (destination,source){

For (var property in source)

destination[property] = source[property];

return destination;

};

* What if the destination is null.
* What if the source is null.

**Rule2:** *Always log the errors to the server.*

Have a look at the below code where the error is logged to the server.

function log(sev, err) {

var img = new image();

img.src = “log.php?sev=” +

encodeURIcomponent(sev) + “msg” +

encodeURIcomponent(msg);

}

log(1, “Something bad happened”);

**Rule3:** *You, not the browser, handle the error.*

The one of the following ways can be used to handle the error by ourselves instead of leaving it to the browser.

try-catch:

try{

operation();

}catch(ex){

log(2, “operation() failed: “ + ex.message);

}

* Thrown error contains extra information
* The errors that are caught are considered to have been handled.

window.onerror:

window.onerror = function(msg, url, line) {

log(1,msg);

return true;

};

* Last stop before the browser responds.
* When the function returns true, it prevents the browser handling the error.

**Error Life cycle**

**Browser error**

**Error**

**try-catch**

**window.onerror**

**Rule4:** *Identify where the errors might occur.*

1. Type coercion error.

Boolean conversion.

if(null) or if(-1) or if(“”) – Always false and never executed.

if(“hi”) or if(-1) or if({}) – Always true and executed.

1. Data type errors.
   1. Often occurs with function arguments.
   2. Typically a symptom of insufficient value checking.
2. Communication errors.
   1. Invalid URL/post data.
      1. Typically long string concatenations.
      2. Don’t forget to use encodeURIComponent() on each part – Not encodeURI()
      3. Make sure parameters are named correctly.
   2. Server response status
      1. 200 is not the only valid status that might be returned. Beware of 304
      2. Any other status means you didn’t get the data.
   3. No network connection
   4. Server response content.
      1. A status of 200/304 is not enough
      2. Server errors often return HTML
      3. If possible set the status to 500.

**Rule5:** *Throw your own errors.*

function sortArray(values){

if(values instanceOf Array){

values.sort(function(a,b){

return b-a;

});

} else{

throw new Error(“sortArray(): invalid arg.”);

}

}

Throw or try-catch?

* Errors should be thrown in the low level parts of the application. – utilities, core libraries, etc,.
* Use try-catch blocks in higher level parts. -Application specific, client side business logic.

**Rule6:** *Distinguish fatal and non-fatal errors.*

* Wont interfere with user’s main tasks.
* Affects only a portion of a page. – Easily disable or ignored.
* Recovery is possible
* A repeat of the action may result in the appropriate result.
* Don’t tell the user it isn’t working unless absolutely necessary.

Fatal errors:

* The application absolutely cannot continue.
* Significantly interferes with user’s ability to be productive.
* Other errors will occur if the application continues.
* Message the user immediately.
* Reload.

Fatal or Non-fatal:

* Don’t allow your code to determine what is and is not fatal.
* The user’s experience comes first.

**Rule7:** *Provide a debug mode.*

* Assign a variable that is globally available.
* Try-catch should rethrow the error.
* Window.onerror should return false.
* Allow the browser to handle the error.