Fixing the SCORM 2004 API Discovery Algorithm

Mike Rustici, Rustici Software, LLC

We all make little mistakes every once in a while, especially when we're working on large projects where the shear quantity of details can be overwhelming. Such is the case with the recently released SCORM 2004 specification. While the new reference model represents a significant leap forward, there are still a few loose ends that need to be tied up. This article will point out some problems with the sample algorithm for finding the API adapter in Figure 3.3.1b of the SCORM 2004 Run-Time Environment Book and also offer an alternative example that you can use in your SCORM conformant content.

First, let's take a look at the existing sample algorithm and where the problems lie. A copy of Figure 3.3.1b from the SCORM Run-Time Environment Book Version 1.3 is included below. I've taken the liberty of adding line numbers for easy reference in later parts of the article.

```
1: var nFindAPITries = 0;
2: var API = null;
3: var maxTries = 500;
4: var APIVersion = "";
5:
6: function ScanForAPI(win)
7: {
8:    while ((win.API_1484_11 == null) && (win.parent != null)
9:          && (win.parent != win))
10:   {
11:         nFindAPITries++;
12:         if (nFindAPITries > maxTries)
13:         {
14:               alert("Error in finding API instance -- too deeply
nested.");
15:               return null;
16:          }
17:         win = win.parent;
18:   }
19:    return win.API_1484_11;
20: }
21:
22: function GetAPI()
23: {
24:    if ((win.parent != null) && (win.parent != win))
25:   {
26:         API = ScanForAPI(win.parent);
27:   }
28:    if ((API == null) && (win.opener != null))
29:   {
30:          API = ScanForAPI(win.opener);
31:          if (API != null)
32:         {
33:               APIVersion = API.version;
34:          }
35:     }
36: }
```

There are two errors in this example and three areas that I believe can be improved upon.

Error #1 – “win” is never declared (line 24)

If you try to run this example code by calling GetAPI(), your browser will conveniently point out to you that the variable “win” is never declared. This error is very easy to fix, by simply adding the line “`var win = window;`” to the first line of the GetAPI function.

Error #2 – There is no “version” property of the API Adapter object (line 33)

Line 33 tries to access the nonexistent version property of the API Adapter. Not being a member of the IEEE LTSC subcommittee that originally wrote this algorithm, I can not speak definitively as to why this line was included, but my best guess is that it comes from an early draft of the P1484.11.1 standard. Regardless of its origins, the API.version call is no longer valid and has been replaced by the data element “_version” that you can access using a GetValue call if needed.

Area for Improvement #1 – Misplaced error message (line 14)

I believe that the error message on line 14 is ill-advised for two reasons. First it is inconsistent. There are two possible ways in which the algorithm may terminate without finding the API Adapter: it can check all possible windows or it can exceed its maximum nesting depth (nFindAPITries > maxTries). The algorithm only provides an error message in one of these instances (when the maximum nesting depth is exceeded). The algorithm should either take responsibility for an error message in both failure scenarios or it should not provide an error message at all.

The second reason is that it may produce an error message when none is warranted. In the scenario where the maximum nesting depth is exceeded when searching the chain of parents, an error message will be displayed despite the fact that the API may still very well be found in the opener window. Granted, it is exceptionally unlikely that the maximum nesting depth will ever be exceeded but unfortunately we programmers have to deal with Murphy’s Law all the time.

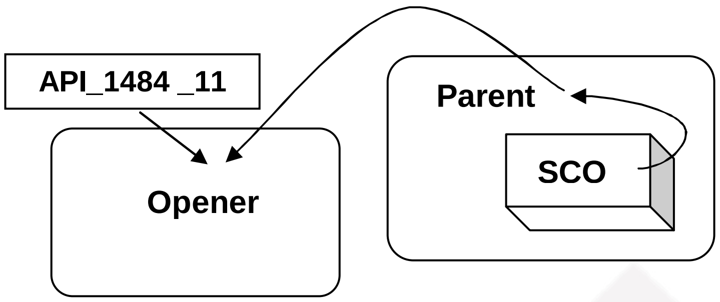
Area for Improvement #2 – Coding style

While what constitutes good coding style is highly subjective, I think we can all agree that global variables should generally be avoided. In the improved example that I provide, I have restructured the code to eliminate unnecessary global variables (note that the one that remains is simply a constant). I also added detailed comments, clarified names and consolidated logic in order to improve the readably of the code and enhance its efficacy as an example.

Area for Improvement #3 – Searching the parent's opener window

This final area for improvement might inspire some discussion. The SCORM specification allows an LMS to launch a SCO either in a frameset or in a new window. In these situations, the API Adapter must reside in the chain of parents of the SCO window, or the chain of parents of the opener of the SCO window.

Historically there have been some LMS's that have deviated slightly from this pattern by launching the SCO in a window, within a frameset, whose opener contains the API Adapter. That is, the API Adapter resides in the base LMS window, this window opens a new popup containing a frameset, this frameset may contain some navigational controls, and also the SCO. Got it? See the diagram below for clarification.



While the technical conformance of this issue can be debated (their side of the argument is that regardless of how many parents the SCO window has, the base window is still the opener), the reality is that as content developers we often have to bend to make things work. That being said, the new example algorithm will search the top window's opener instead of searching only the current window's opener. This change will allow your content to find the API in the above scenario and will still function correctly when the content is opened directly in a new window in which case the top window will simply evaluate to the current window.

This change will also allow you to use this algorithm from within any child window of you content, not just the top-level window. Currently, many SCO developers make the mistake of using the default ADL algorithm in deeply nested windows. While this placement functions properly when the API Adapter is in a parent window, it will break down when the API Adapter is in the SCO's opener window.

The New and Improved Example

So now that we have seen the problems with the existing example, let's take a look at the new and improved example. You can download a script file containing this improved example from our website at www.scorm.com.

```
var MAX_PARENTS_TO_SEARCH = 500;
```

```
/*
ScanParentsForApi
-Searches all the parents of a given window until
 it finds an object named "API_1484_11". If an
 object of that name is found, a reference to it
 is returned. Otherwise, this function returns null.
*/
function ScanParentsForApi(win)
{

      /*
      Establish an outrageously high maximum number of
      parent windows that we are will to search as a
      safe guard against an infinite loop. This is
      probably not strictly necessary, but different
      browsers can do funny things with undefined objects.
      */
      var nParentsSearched = 0;

      /*
      Search each parent window until we either:
             -find the API,
             -encounter a window with no parent (parent is null
                        or the same as the current window)
             -or, have reached our maximum nesting threshold
      */
      while ( (win.API_1484_11 == null) &&
                  (win.parent != null) && (win.parent != win) &&
                  (nParentsSearched <= MAX_PARENTS_TO_SEARCH)
              )
      {

            nParentsSearched++;
            win = win.parent;
      }

      /*
      If the API doesn't exist in the window we stopped looping on,
      then this will return null.
      */
      return win.API_1484_11;
}


/*
GetAPI
-Searches all parent and opener windows relative to the
 current window for the SCORM 2004 API Adapter.
 Returns a reference to the API Adapter if found or null
 otherwise.
*/
function GetApi()
{
      var API = null;

      //Search all the parents of the current window if there are any
      if ((window.parent != null) && (window.parent != window))
      {
            API = ScanParentsForApi(window.parent);
      }

      /*
      If we didn't find the API in this window's chain of parents,
      then search all the parents of the opener window if there is one
      */
      if ((API == null) && (window.top.opener != null))
      {
            API = ScanParentsForApi(window.top.opener);
      }

      return API;
}

/********************************************************************
Sample usage
********************************************************************/
var objScormApi;

objScormApi = GetApi();

//if we didn't find the API, alert the user
//note - you might want to disable future attempts at
//calling API functions at this point
if (objScormApi == null){
      alert("Error finding API instance");
}

objScormApi.Initialize("");
```

To use the new code, simply call the GetApi function. Once the call has returned, compare the return value with null to ensure that the API Adapter was found. If the API Adapter was not found, it is a good idea to display an error to the user and then disable future attempts at accessing the API. As a best practice, I would recommend that you only search for the API Adapter once and then maintain a constant reference to it in memory. As a corollary, if the API Adapter is not found, you should only display an error message once and then allow your content to degrade gracefully by disabling future attempts to access the API Adapter.

In this article we learned about several problems with and areas for improving the standard SCORM 2004 API Adapter discovery algorithm. Feel free to use the improved example in all of your SCORM conformant content.

Author's Note – This article was written in March of 2004 and applies to the SCORM Run-Time Environment book released on January 30, 2004. ADL is scheduled to issue a maintenance release to this book during the summer of 2004 in which the above mentioned issues are scheduled to be addressed.

About the Author

Mike Rustici is a software consultant who specializes in helping clients conform to the SCORM and other learning standards. A veteran of the online training industry, Mike has a unique blend of both LMS development and content production experience. As an active member of the ADL SCORM community, he helps influence both the future direction of standards development and the interpretation of existing standards. Rustici Software, LLC has helped numerous clients throughout industry, government and academia. To learn more about Rustici Software, LLC, please visit http://www.scorm.com.