



***Got SCORM? A Brief Technical Overview That Answers  
the Question 'What is SCORM?'***

**Brian Kleeman**  
Technology Director and  
Lead Software Developer

**ICS Learning Group  
650 Ritchie Highway  
Severna Park, MD 21146**

**410-975-9440**  
**[www.icslearninggroup.com](http://www.icslearninggroup.com)**

## ***Disclaimer***

ICS Learning Group makes no representations or warranties with respect to the contents or use of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. ICS Learning Group reserves the right to revise this publication and to make changes to its content at any time, without obligation to notify any person or entity of such revisions or changes.

Further, ICS Learning Group makes no representations or warranties with respect to any ICS Learning Group product, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. ICS Learning Group reserves the right to make changes to any and all parts of ICS Learning Group products at any time, without obligation to notify any person or entity of such changes.

## ***Trademarks***

Inquisiq, Inquisiq EX, and ICS Learning Group are registered trademarks of Interactive Communications Solutions Group, Inc.

Other brands and their products are trademarks or registered trademarks of their respective holders.

## Introduction

When I started researching SCORM in preparation for developing content and implementing an LMS, what I remember most was my frustration at the lack of any concise, hands-on information that was available about SCORM. Advanced Distributed Learning (ADL) has the complete specification (all 400+ pages of it) readily available on their site, but I was looking for a quick, “get your hands dirty” explanation that skipped all the mumbo-jumbo and would let me dive in and start trying things out.

Hopefully, you’ve found this whitepaper early in your search and avoided that frustration. My purpose is to give you a “real world” explanation of SCORM that you can actually use to begin understanding the details of what you’ll find in the specification itself. I’ll skip the e-learning history, theory and other vague ramblings as I’m sure you can find an abundance of it elsewhere with relative ease.

## Overview

The SCORM specification, developed by Advanced Distributed Learning (ADL - <http://www.adlnet.org>), is a set of rules that learning management systems (LMS) and learning content follow in order to be compatible with each other. This theoretically allows the content to be loaded into, launched and tracked by any learning management system using a common rule set. Imagine your typical kitchen toaster and its electric plug that you plug-in to your wall socket. That same wall socket can have a blender, electric can opener or coffee maker plugged into it because all those appliances, along with the socket itself, comply with a set of rules dictating plugs and sockets. Things such as the number of prongs, shape and dimensions of the prongs, polarity and voltage have been published as a standard that manufacturers follow to insure that their plugs and sockets all work together. The SCORM specification is no different. Your learning management system is the wall socket. Your content is the toaster, blender, coffee maker or whatever else you plug into that socket.

## Introducing the SCO...

E-Learning content is delivered as a single unit called a Sharable Content Object or SCO. SCOs are independent, self-contained, transportable packages that represent the lowest level of granularity that is tracked by a learning management system. *In other words, SCOs are mini-applications that when launched from an LMS report, among other things, one score and pass/fail status.* Your SCOs should contain all the necessary files required to function, have or require no knowledge of any other SCO or external information, and as a matter of design, should remain relatively small such that they can be easily reused.



We generally design our SCOs (which we also refer to as lessons) to be able to be completed by a learner in less than 15 minutes. We’ve found that this avoids information overload and gives the learner well spaced stopping points.

And since the spec indicates that SCOs shall be transportable, you could infer that they should also be platform independent, although this is not explicitly stated. Server dependencies such as Coldfusion, Perl, ASP or server-based databases are problematic as a particular LMS might be running on a server that does not have those services available. So unless you are working in a tightly controlled internal environment, your SCOs should only be using client-side technologies such as HTML, Flash and Javascript.

### ***...and the Content Package***

Now before we get into the details of the SCO, realize that we must be able to transport it – even if that only means getting it from our desktop computer where we created it to the server where our LMS is hosted. To do so, we use a Content Package. Going back to our wall socket analogy, imagine that the content package is the box in which the toaster is packed and transported. In reality, it is simply a single compressed file containing all the files (html, images, flash swf, etc) necessary for the SCO to function.

The content package may contain more than one SCO (a toaster and coffee-maker perhaps) and a single manifest file which describes the contents of the package. The LMS will use the manifest file to properly find and import each of the SCOs.

### ***Web-Based and Script Enabled***

While there is no specific requirement that learning management systems be web-based, SCOs must be. This requirement gives developers a common platform (the web browser) to design and develop their content to function within, thereby removing unnecessary barriers to compatibility and transportability. Of course, even with the current maturity of web-browsers, they are not all created equal and SCORM makes no mention of a specification to be followed to ensure web-browser compatibility. Compliance with the latest W3C standards however, is a safe course of action to ensure compatibility with a wide range of current web-browsers.

And in order for a SCO to communicate data to and from the LMS as well as potentially provide interactivity and quizzing functionality, it's going to require more than just client-side HTML. SCORM specifies that SCOs use an ECMA Standards compliant scripting language to communicate with the LMS. The most notable ECMA compliant language is one you've probably heard of - JavaScript.

### ***SCO and Content Package Summary***

So what we've covered regarding Sharable Content Objects can be summarized as follows:

- SCOs are mini-applications containing the content itself.
- SCOs are self-contained and transportable – all required files are included and compressed into a single content package.
- Content packages contain a single manifest file and may contain more than one SCO.
- SCOs are web-based and script-enabled – consisting of HTML, Javascript and any other client-side technology (images, Macromedia Flash, etc).

- SCOs are the smallest level of granularity tracked by an LMS – they report, among other things, only one score and status.

## The Details

Now that you understand some of the high-level layout, we'll explore the fundamental details to understanding SCORM. Realize that this is by no means an exhaustive exploration, but it should provide a good foundation of knowledge. Be sure to download and read the complete SCORM specification as there are many details that will not be mentioned here.

### *The Content Package and Manifest File*

As I have mentioned, for a SCO to be uploaded to the LMS, it must be contained in a Content Package - a single compressed file conforming to the Process Interchange Format (PIF) specification. In most cases, this is simply a .ZIP file and may contain multiple SCOs.

The Content Package must also contain a Manifest File – an XML file containing information about the included SCOs and their organization. By reading the manifest file, the LMS can gather information about the SCOs that are contained in the package and will be able to launch them when appropriate.



The Manifest File must be located in the root of the Content Package and named 'imsmanifest.xml'. The remaining file structure of the content package is completely up to you, even when including multiple SCOs, as long as the references contained in the manifest file correctly identify the location of the launching files, metadata files or any other files that the LMS might need to find within the package.

Here's an example of a very basic, yet fully compliant "hello world" manifest file. Note that a fair amount of optional information not covered here is usually included in a typical manifest file. For complete details be sure to refer to Section 3, Content Package Conformance Requirements of the SCORM 2004 (1.3) Conformance Requirements (CR).

```
<?xml version="1.0" encoding="utf-8" ?>
<manifest identifier="ims_hello" version="1.0"
  xmlns="http://www.imsproject.org/xsd/imsscp_rootv1p1p2"
  xmlns:adlcp="http://www.adlnet.org/xsd/adlcp_rootv1p2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.imsproject.org/xsd/imsscp_rootv1p1p2_imsscp_rootv1p1p2.xsd
    http://www.imsglobal.org/xsd/imsmd_rootv1p2p1_imsmd_rootv1p2p1.xsd
    http://www.adlnet.org/xsd/adlcp_rootv1p2_adlcp_rootv1p2.xsd">
  <metadata>
    <schema>ADL SCORM</schema>
    <schemaversion>CAM 1.3</schemaversion>
  </metadata>
  <organizations default="crs_hello">
    <organization identifier="crs_hello">
      <title>Hello World Course</title>
      <item identifier="sco_hello" identifierref="sco_hello_ref">
        <title>Hello World Lesson</title>
      </item>
    </organization>
  </organizations>
  <resources>
    <resource identifier="sco_hello_ref">
```

```
        type="webcontent"
        href="index.htm"
        adlcp:scormtype="sco">
    </resource>
</resources>
</manifest>
```

Notice that the basic structure is a `manifest` node containing a `metadata` node, an `organizations` node and a `resources` node.

## The Metadata Node

The required elements of this node that we have included specify the schema and schema version. Our example indicates that this content package uses the ADL SCORM schema and is compliant to the Content Aggregation Model of the SCORM 2004 (1.3) specification. Optionally, this node may also contain metadata containing extensive information about the content package including keywords, copyright information and more. This optional metadata may be included using a `lom` node or in an external XML file whose location would be referenced by an `adlcp:location` node. In either case, the `lom` or `adlcp:location` node would be located within the `metadata` node.



The meta-data structure, whether in-line or external is defined in Section 5 of the SCORM 2004 (1.3) Conformance Requirements (CR). Note that several other areas of the manifest also use this setup for referencing meta-data.

## The Organizations Node

The primary layout and relationship of the contents of our package is contained in the `organizations` node. Notice however, that information about the assets themselves such as file location are not included here.

```
<organizations default="crs_hello">
  <organization identifier="crs_hello">
```

You'll notice a single organization in this node, although any number is allowed. Each organization that is listed in this node can be thought of as a course once it's uploaded into the LMS.



The SCORM specification does not require that an LMS recognize and import the organization structure itself – only that it be able to import and launch SCOs and assets that are defined within the `resources` node and referenced by item nodes within in the `organizations` node.

Each organization listed must contain a `title` node to indicate the title of the organization or course.

```
<title>Hello World Course</title>
```

We have titled our sample course "Hello World Course." Once imported, the LMS will use this as the display title of the course.

Each organization may contain any number of `item` nodes which, in turn, may also contain any number of `item` nodes. There is no limit to the nesting of these nodes. The lowest level of `item` nodes represents the SCOs or assets contained in the content package and will each include an attribute `identifierref`.

```
<item identifier="sco_hello" identifierref="sco_hello_ref">
```



An Asset is simply a file or group of files that can be launched by the LMS. Assets differ from SCOs in that they do not communicate with the LMS in any way.

Our sample's `identifierref` value is "sco\_hello\_ref" which references a `resource` node that we will discuss shortly. An individual `resource` node may be referenced by more than one `item` node throughout the `organizations` node, even from within different organizations. When this occurs, the `identifierref` attribute for all instances should be the same.

When present, upper layers of the `item` node nesting represent groupings or modules within the organization as shown by this example not contained in our sample:

```
<organization identifier="crs_hello">
  <title>Hello World Course</title>
  <item identifier="module1">
    <title>Module One</title>
    <item identifier="sco1" identifierref="scolref">
      <title>SCO One Title</title>
    </item>
  </item>
</organization>
```

Most manifest files that you see coming from authoring programs such as Macromedia Captivate and Articulate Presenter will not nest the `item` nodes into modules, but rather contain only one `item` node (representing the SCO) within the organization.

Each `item` node representing our SCOs and assets must contain a `title` node to indicate the title of the resource itself.

```
<title>Hello World Lesson</title>
```

We have titled our SCO "Hello World Lesson" and the LMS will use this as the displayed title of the lesson once imported.

## The Resources Node

Whereas the `organizations` node defined the layout and relationship of the content, the `resources` node is where the actual content information is located. Each SCO or Asset that is contained within the content package will have a corresponding `resource` node within this node.

```
<resource
  identifier="sco_hello_ref"
```

```
type="webcontent"  
href="index.htm"  
adlcp:scormtype="sco">
```

The `identifier` attribute of our resource contains the value `"sco_hello_ref"`, which will correspond to the `identifierref`'s value in our `item` node that is in the `organizations` node above.

The `href` attribute lets the LMS know where the launch file for this resource is located and the `adlcp:scormtype` attribute indicates that this resource is a SCO rather than simply an Asset. So in our sample, the `index.htm` file located in the root of the content package as indicated here, is the launch file for the SCO that is titled "Hello World Lesson" as listed in the `organizations` node and referenced using the `"sco_hello_ref"` identifier.

The result of all of this is that when an LMS imports a content package containing our sample manifest it will, at a minimum, create a new lesson, title it "Hello World Lesson" and upon launch execute the file "index.htm" that is located in the root of the content package. Optionally, the LMS may also create a new course, title it "Hello World Course" and include our "Hello World Lesson" in it. In either case, our LMS has successfully imported our SCO and has the capability to launch it.



For a complete explanation of the structure and elements included in the manifest, refer to Section 3 of the SCORM 2004 (1.3) Conformance Requirements (CR).

## A Communications Channel - The API

Once it's been launched, a SCO and the LMS need a communication channel through which to pass data. This is done through an object known as an *API (Application Programming Interface)* that is the conduit for all of the SCO to LMS and LMS to SCO communication.

The technology that the LMS vendor chooses to implement the API is completely up to them - no recommendations are made within the SCORM specification as long as the API is exposed to the SCO correctly and implements the methods and data structure required. We have seen APIs developed using Java applets, ActiveX controls and even pure JavaScript.

The SCO will be opened by the LMS in either a new browser window or framed within a page in the LMS browser window. Before communication can occur, the SCO must find the API and make contact.



In most cases, the API is visible in the HTML code of the LMS window as an `object`, `embed` or `applet` tag.

The SCORM specification indicates locations where the SCO should look for the API, requires that the LMS expose the API in one of those locations and that it be a Document Object Model (DOM) object named `"API_1484_11"` or `"API"` depending on

the SCORM version you are following. Since the SCO is launched in either a new browser window or a frame within the LMS window, it will use a very simple process such as: look in my parent window (if framed) or in the window that opened me (if a new window) and see if there is an object named "API\_1484\_11" (or "API"). If the API is not found, the SCO can expand its search by looking in the parent of the window that opened it, the parent of its parent, and so on according to the spec.



Section 3.2.1 of the SCORM 2004 (1.3) Run Time Environment (RTE) outlines the locations and process that the SCO should use to find the API.

Once the SCO finds the API, it can invoke methods of the API to send data to and receive data from the LMS. If the API is not found, the SCO should alert the user that the connection to the LMS failed and no communication will occur.



The ADL has made an "API wrapper" publicly available for use when developing SCOs. This wrapper is a file that you can include with your SCO that contains pre-written javascript functions for finding and accessing the API as well as for sending and receiving data. The wrapper is available on the ADL's website at <http://www.adlnet.org>

## Data Transfer

Once the SCO has found the API, they must both speak the same language if any communication is to occur. The SCORM specification has defined a small set of methods that must exist in the API and be available for the SCO to use. The methods are accessed by the SCO via JavaScript code with the syntax of *objectname.methodname(argument(s))* where *objectname* references the API itself, *methodname* is the method being used (the API methods are explained below) and *argument(s)* are the data passed to the method. In all cases, the SCO initiates interactions and data transfer by invoking these methods.

### ***SCORM 2004 (1.3) API Methods:***

*Initialize* – Initializes communication with the LMS. No other API methods should be called by the SCO until *Initialize* has been successfully called.

```
returnValue = API_1484_11.Initialize("");
```

When completed, the resulting `returnValue` will contain "true" if the method was successful, "false" if it was not.

*Terminate* – Terminates communication with the LMS. No other API methods should be called by the SCO after *Terminate* has been successfully called.

```
returnValue = API_1484_11.Terminate("");
```

When completed, the resulting `returnValue` will contain "true" if the method was successful, "false" if it was not.

Commit – Saves the data that has been sent to the LMS via SetValue calls. If a SCO exits without invoking *Commit*, none of the learner’s data is saved to the LMS. *Commit* is implicitly invoked by the API when *Terminate* is called.

```
returnValue = API_1484_11.Commit("");
```

When completed, the resulting `returnValue` will contain “true” if the method was successful, “false” if it was not.

GetValue – Retrieves data from the LMS for use in the SCO. The SCO must pass the data element that it is requesting as an argument.

```
returnValue = API_1484_11.GetValue("cmi.score.raw");
```

When completed, the resulting `returnValue` will contain the score that is retrieved from the LMS.

SetValue – Passes data from the SCO to the LMS. The data is retained and may be retrieved during the user session, but is not saved to the LMS until *Commit* is invoked. The SCO must indicate the data element and its value that is to be saved as arguments.

```
returnValue = API_1484_11.SetValue("cmi.score.raw", "98");
```

When completed, the resulting `returnValue` will contain “true” if the method was successful, “false” if it was not.

GetLastError – Retrieves the last numeric error code that occurred in the API as a result of invoking these methods.

```
returnValue = API_1484_11.GetLastError();
```

When completed, the resulting `returnValue` will contain the code corresponding to the last error that occurred.

GetErrorString – Retrieves the text description corresponding to the error code provided.

```
returnValue = API_1484_11.GetErrorString(errorCode);  
returnValue = API_1484_11.GetErrorString(122);
```

When completed in either case, the resulting `returnValue` will contain a text description of the error corresponding to the `errorCode` that was provided.

GetDiagnostic – Exists for LMS specific use. Returns a diagnostic text description based on the parameter that is passed as an argument.

```
returnValue = API.GetDiagnostic("diagnostic text");
```

When completed, the resulting `returnValue` will contain the text of the diagnostic information.



A SCORM-compliant SCO is only required to call the *Initialize* and *Terminate* methods – all other method calls are optional. This will not result in any status or scores being passed, but will notify the LMS that the SCO was successfully launched and exited.

Using the *GetValue* and *SetValue* methods, the SCO is able to send and retrieve all the necessary data for effective tracking to and from the LMS. Some commonly used data elements include `learner_id`, `learner_name`, `score`, `completion_status` and `suspend_data`. The example that follows will demonstrate proper usage of a few of these elements.



A complete listing of the data model elements and their descriptions can be found in Section 4.2 of the SCORM Run-Time Environment (RTE).

## An Example

So what you will find when looking at the code of a SCO is some process that is executed at launch, usually found in the `onload` event of the `body` tag, that finds and initializes the API. In some cases the SCO will then request and load basic information using *GetValue*, such as the learner's name and id. Additionally, the SCO may attempt to load previous session data such as score, progress status or learner responses that may be necessary to continue the learner's interaction with the SCO. From that point, SCO interactivity and functionality such as navigation and quizzing will likely operate independently of any SCORM-related functions. At certain points though, result data is passed to the LMS (using *SetValue* and possibly *Commit*) and upon completion of the SCO, perhaps within the functionality of an Exit button, the *Terminate* method is called.

Let's take a look at the following HTML and Javascript code as an example. Note that the *getAPI* function used in the *doLMSInitialize* function would be included in the `apiwrapper.js` script file that is referenced by the script tag in the header. This function would follow the process discussed earlier to search for and return the API from the opener window or parent frames. While this process is too detailed to completely explain here, the api wrapper that is available from the ADL contains this functionality so it is not necessary to develop your own from scratch.

```
<html>
<head>
<script type="text/javascript" src="apiwrapper.js"></script>
<script type="text/javascript">
var API;
var learnerName;
function doLMSInitialize(){
    //find and returns the API starting with this window
    API = getAPI();

    API.Initialize("");
    learnerName = API.GetValue("cmi.learner_name");
}
```

```

        //write the welcome message
        var welcomeDiv = document.getElementById("welcome");
        welcomeDiv.innerHTML = "Welcome " + learnerName + "!";
    }

    function postAnswer(str){
        if (str.toLowerCase() == "blue"){
            API.SetValue("cmi.score.raw", "100");
            API.SetValue("cmi.score.scaled", "1");
            API.SetValue("cmi.success_status", "passed");
            API.SetValue("cmi.completion_status", "completed");
            alert("Correct!");
        } else {
            API.SetValue("cmi.score.raw", "0");
            API.SetValue("cmi.score.scaled", "0");
            API.SetValue("cmi.success_status", "failed");
            API.SetValue("cmi.completion_status", "completed");
            alert("Incorrect. Try Again.");
        }
    }
}
</script>
<title>SCO Example</title>
<body onload="doLMSInitialize();">
<div id="welcome"></div>
<form name="answerForm">
What color is the sky? <input type="text" name="answer"><br />

<input
    type="button"
    name="submit"
    onclick="postAnswer(document.answerForm.answer.value);"
    value="Submit">

</form>
<input type="button" onclick="API.Terminate('');" value="Exit">
</body>
</html>

```

Let's briefly step through this code to see what is actually happening. On page load, the *doLMSInitialize* function is called which uses the *getAPI* function to search for and return the API. The API is initialized, the learner's name is retrieved and a welcome message is displayed in the *div* element whose *id* is "welcome."

```

function doLMSInitialize(){
    //find and returns the API starting with this window
    API = getAPI();

    API.Initialize("");
    learnerName = API.GetValue("cmi.learner_name");

    //write the welcome message
    var welcomeDiv = document.getElementById("welcome");
    welcomeDiv.innerHTML = "Welcome " + learnerName + "!";
}

```

The question is displayed using a standard HTML form. Upon clicking the Submit button, the learner's response is passed to the *postAnswer* function.

```

<form name="answerForm">
What color is the sky? <input type="text" name="answer"><br />

<input
    type="button"
    name="submit"
    onclick="postAnswer(document.answerForm.answer.value);"
    value="Submit">

```

```
</form>
```

The *postAnswer* function evaluates the response, alerts the learner and sets the appropriate SCORM values using the API's *SetValue* method.

```
function postAnswer(str){
  if (str.toLowerCase() == "blue"){
    API.SetValue("cmi.score.raw", "100");
    API.SetValue("cmi.score.scaled", "1");
    API.SetValue("cmi.success_status", "passed");
    API.SetValue("cmi.completion_status", "completed");
    alert("Correct!");
  } else {
    API.SetValue("cmi.score.raw", "0");
    API.SetValue("cmi.score.scaled", "0");
    API.SetValue("cmi.success_status", "failed");
    API.SetValue("cmi.completion_status", "completed");
    alert("Incorrect. Try Again.");
  }
}
```

Finally, the learner clicks the Exit button which invokes the API's *Terminate* method.

```
<input type="button" onclick="API.Terminate(''); " value="Exit">
```

What this example should illustrate for you is that a SCO is coded very much like any other interactive lesson with the occasional addition of method calls to the API when data is required or ready to be posted.

You can download this SCO and content package including the example manifest file at <http://www.icslearninggroup.com/resources/helloworld.zip>. This package is a functional SCORM 1.3 (2004) package that can be imported into a SCORM compliant learning management system.

## Conclusion

So we've examined the SCO and its general operation, the API and how it provides the communications portal between the LMS and the SCO and the manifest file and how it facilitates the importation of the SCO into the LMS. In other words, we've only scratched the surface. To fully understand SCORM and its functionality, you should download the full specification from the ADL's website and get familiar with it. Begin working with our sample content package and study as many other SCORM compliant SCOs as you can. The code is all client-side, so you will be able to open it with a basic text editor such as Notepad.

For more information on SCORM, learning management systems, SCORM compliant authoring tools, or to sign up for a free trial of Inquisiq EX Learning Management System where you can upload and test your SCORM compliant SCOs, visit us on the web at <http://www.icslearninggroup.com>.