SCORM Developer's Toolkit

Web site: www.e-learningconsulting.com
E-mail: info@e-learningconsulting.com

E LEARNING CONSULTING

Table of Contents

About This Guide

Who Should Use the SCORM Developer's Toolkit

The toolkit is designed for e-learning course developers who want to create SCORM-based courses using with HTML/JavaScript or Flash. You will find the toolkit very easy to use if you are a web or Flash developer who can use JavaScript or ActionScript to create interactive web pages or Flash movies. The toolkit will let you add the SCORM portion of your course quickly and with little fuss.

The toolkit is designed to save you time. The toolkit contains well documented and tested JavaScript functions that correctly implement SCORM 1.2 and SCORM 2004 communications with the LMS. The same toolkit functions can be used for both SCORM 1.2 and SCORM 2004 communications!

The toolkit is not for everyone. You will have trouble using the toolkit if you have never used JavaScript (or ActionScript for Flash developers).

How to Use This Guide

Read the Anatomy of a SCORM Course section to get a general understanding of SCORM. Read the remaining sections to learn how to use the SCORM Developer's toolkit.

Typographic Conventions

Bold Arial Font is used to identify user the name of files, JavaScript functions and parameters to those functions.

Anatomy of a SCORM Course

A SCORM course[1] is a collection of one or more Sharable Content Objects (SCOs). A SCO can provide any type of self-paced learning activity. For example, a SCO could provide a tutorial, a quiz, a simulation or a test. The Learning Management System (LMS) launches the SCOs within a course. A SCO must contain at least one HTML page that contains JavaScript. SCOs can be small (a single page) or huge (thousands of pages). You will decide what size SCO works best for you based on the instructional requirements of your course. A SCO can contain any type of content that can be delivered in a browser (HTML text, graphics, animation, audio and video). A SCO can use plug-ins. The design and content of the SCO is entirely up to you.

The SCORM standard defines the SCORM Runtime API that lets a SCO communicate with the LMS. The LMS provides the SCORM Runtime API for the SCO to use when the SCO is launched. The SCORM API lets the SCO save and restore state information. State information can include information important to the SCO such as the last page viewed and the learner's response to questions. Saving and restoring state information lets a learner:

1. Launch the SCO and interact with it.
2. Stop the SCO (close the browser window or navigate away from the SCO).
3. Relaunch the SCO. The relaunched SCO can then retrieve the state information from the LMS and then use that state information to return the learner to the last viewed page and restore all of his interactions with the SCO.

SCORM refers to the launching of a SCO as a session. Every time a learner launches a SCO, a new session is started. A learner can complete a SCO in one or more sessions. The SCO decides when the learner has done enough to complete an attempt on the SCO. This attempt can be spread out over one or more sessions.

The SCORM Runtime API also allows the SCO to record information about the learner's actions with the SCO. This information can include:

[1] The SCORM specification does not actually use the word "course" to define a collection of SCOs. Instead it refers to a collection of SCOs as a "content aggregation". Most developers are more comfortable with the word "course" so we use it throughout the guide.

- If the learner has completed the attempt in this session
- If the learner actions with the SCO were judged as passed or failed
- An overall score
- The results of individual interactions
- The completion of objectives
- The time of each session
- Other information such as the learner's comments

Toolkit Overview

The SCORM Developer's Toolkit helps you create courses that conform to the SCORM[2] standard. The toolkit contains JavaScript functions, HTML/JavaScript samples, a Flash sample and this guide. The toolkit provides a rich set of JavaScript functions that make it easy to work with SCORM. This lets you spend your time thinking about the unique functionality of your SCO. The toolkit JavaScript functions will help you:

1. Develop SCORM-based courses from scratch
2. Modify existing courses so they will support SCORM
3. Create HTML/JavaScript courses that will work with or without framesets.
4. Create Flash courses.

To create SCOs with the toolkit, you must be familiar with HTML and JavaScript. You must be familiar with ActionScript if you would like to create a SCO with Flash.

Generally a SCO behaves in this way (your SCO will use the functions provided by the toolkit to perform these actions):

1. The SCO initializes the communications session with the LMS.
2. Records the start time of the SCO.
3. Detects if this is the first time the learner has launched the SCO (the first session). If this is the first launch, the SCO shows the initial content of the SCO. If the learner has already launched the SCO (has already completed at least one session), the SCO gets the bookmark (generally this is the last page viewed) and other state information it has saved in previous sessions.
4. Provides the instructional interactions required by the SCO. The instructional interactions could include the presentation of a tutorial, delivering a quiz or test, interacting with the learner through a simulation, etc.

[2] The SCORM Developer's Toolkit supports SCORM 1.2 and SCORM 2004. All references to SCORM in this document refer to SCORM 1.2 and SCORM 2004.

5. The SCO reports information to the LMS at appropriate times. This information can include:

   - Set a new bookmark when the learner visits a new part of the SCO (for example, visits a new HTML page or navigates to a section within a Flash movie).
   - Set the state information to keep track of the learner's actions. For example, the state information might be updates to keep track of the learner's response to a question.
   - Tell the LMS how the learner performed on a specific question. For example, the SCO can tell the LMS that a learner completed a question with a specific identifier. This question was a true-false type question. The learner responded with true but the correct answer was false so the learner was incorrect.
   - Determine if the learner has completed the SCO.
   - Determine if the learner has passed or failed.
   - Record an overall score for the SCO.
   - Get and set other SCORM data items such as the completion of objectives within the SCO.

6. Record the time to complete this session.
7. The SCO can set the information described at any time during the session. The SCO must communicate this information before the SCO is unloaded (browser closed or the browser is loaded with another SCO).
8. Ends the communication session.

Toolkit JavaScript Functions

The toolkit's JavaScript functions are in **sco_api.js**. You will include **sco_api.js** in all of your SCOs so you can call these JavaScript functions from your SCO's HTML pages and Flash movies.

Your SCO must call the JavaScript functions to initiate and terminate the SCO communications session. Your SCO may call the other JavaScript functions as needed. For example, you can decide whether or not your SCO will call the **setScore()** function to record a score for your SCO.

The JavaScript functions fit within these categories:

- **Session** – initiate and terminate the SCORM communication session
- **Launch conditions** – functions that provide information about the launch of the SCO
- **Time** – functions to manage the session time
- **State management** – functions to manage the bookmark and other state data
- **Completion** – functions to handle the completion of the SCO
- **Pass/Fail** – functions to set/get the pass/fail status of the SCO
- **Score** – functions to set/get the score of the SCO
- **Interaction** – functions to set/get interactions
- **Objective** – functions to set/get objectives
- **Type of communications** – functions that let your SCORM identify the type of communications available with the LMS
- **Lower level** – lower level functions to directly call the SCORM 1.2 and SCORM 2004 Runtime API provided by the LMS

Session Functions

Your SCO needs to initiate and terminate the communication session with the LMS. The initiation and termination of the session must be done every time your SCO is launched. You will usually initialization the session as soon as your SCO is loaded. You will usually terminate the session when your SCO is unloaded or you have finished communicating with the LMS for this session. The 5 samples show ways to call these session functions.

Your SCO should also tell the LMS if the learner will return in a follow-on session to complete the attempt on the SCO.

Initialize a Session

You must initialize the SCORM communications session every time your SCO is launched. You must initialize the SCORM session before you call any other functions in the toolkit.

function initCommunications()

Parameters: none

Returns: nothing

Example

```
<script language="javascript" type="text/javascript">
function initSCO() {
        /* tell SCORM API we have started the SCO session */
        initCommunications();

        … more initialization for this SCO …
}
</script>

<body onload="initSCO()" onbeforeunload="termSCO()" onUnload="termSCO")>
```

SCORM Version	Action
SCORM 1.2	Calls API.LMSInitialize("")
SCORM 2004	Calls API_1484_11.Initialize("")

Terminate a Session

Your SCO must terminate the SCORM communications session before it is unloaded. You cannot call any other functions in the toolkit after you terminate the SCORM session.

function termCommunications()

Parameters: none

Returns: nothing

Example

```
<script language="javascript" type="text/javascript">
function termSCO() {
        /* see if we have already called this function */
        if (!_bTerminated) {
                /* we have not, so set this var to make sure we do this only once */
                _bTerminated = true;

                /* record the session time */
                setSessionTime(_timeSessionStart);

                /* tell SCORM we are done with this SCO session */
                termCommunications();
        }
}</script>

<body onload="initSCO()" onbeforeunload="termSCO()" onUnload="termSCO")>
```

SCORM Version	Action
SCORM 1.2	Calls API.LMSFinish("")
SCORM 2004	Calls API_1484_11.Terminate("")

Tell the LMS the Learner Will Return in another Session

A learner can take several sessions to complete his/her attempt on a SCO. Your SCO should tell the LMS if the learner will return in another session to complete the SCO.

The LMS will only save the bookmark, suspend data and other SCORM state data (such as interactions in SCORM 2004) between sessions if you explicitly tell the LMS that the learner will return.

function learnerWillReturn(bWillReturn)

Parameters: bWillReturn – true if the learner will return to the SCO in another session and the state data should be saved for the next session, else false

Returns: nothing

Example

```
/* the attempt on this SCO is not complete, so tell the LMS that the learner will return
and we want it to retain the bookmark, suspend data and other state data so we will
have it available in the next session */
learnerWillReturn(true);
```

SCORM Version	Action
SCORM 1.2	Sets the value of API.cmi.core.exit
SCORM 2004	Sets the value of API_1484_11.cmi.exit

Launch Condition Functions

Your may want to have your SCO work in different ways depending on the launch conditions. For example, if your SCO has been launched before, you may want to restore the bookmark and automatically navigate to that bookmark within your SCO. You launch conditions provide these functions:

- If this is the first launch of the SCO by the learner
- The launch data available to the SCO
- If this SCO has been launched for credit
- If this SCO has been launched in normal, review or browse mode

Is This the First Launch of the SCO

You may want your SCO to behave differently if this is the first time the learner has launched the SCO versus if this is a follow-on launch of the SCO. If the SCO has been launched before, you may want to retrieve the bookmark and other state information that your SCO set in previous sessions.

function isFirstLaunch()

Parameters: none

Returns: **true** if this is the first launch of the SCO, else **false**

Example

```
/* see if this is the first launch of the SCO by this learner */
if (isFirstLaunch()) {
                /* it is, set the status to incomplete so the LMS knows the learner is not
done with this SCO yet */
        setCompletionStatus("incomplete");

        /* we will need to get the bookmark in the next session so tell the LMS that the
learner may return in the future */
        learnerWillRelaunch(true);

        /* go to the first page of the course */
        gotoPage(0);
} else {
        /* not the first launch, so we can get the bookmark (the bookmark was set in a
previous session) */
        var sBookmark = getBookmark();

        /* go to the bookmarked page */
        gotoPage(sBookmark-0);
}
```

SCORM Version	Action
SCORM 1.2	Gets the value of API.cmi.core.entry. Returns true is the value = "ab-initio"
SCORM 2004	Gets the value of API_1484_11.cmi.entry. Returns true is the value = "ab-initio"

Get the Launch Data

The imsmanifest.xml file describes a SCORM course. This file can include a tag that contains the launch data for the SCO. If this tag is missing, getLaunchData() will return "" (an empty string). The name of the tag depends on the version of SCORM.

The tag is <adlcp:datafromlms> for SCORM 1.2. The tag is <adlcp:dataFromLMS> for SCORM 2004. Your SCO can use this launch data.

function getLaunchData()

Parameters: none

Returns: A string containing the launch data contained in the imsmanifest.xml file for this SCO

Example

```
/* get the launch data */
var sLaunchData = getLaunchData();
```

SCORM Version	Action
SCORM 1.2	Returns the value of API.cmi.launch_data
SCORM 2004	Returns the value of API_1484_11.cmi.launch_data

Get the Credit/No-Credit Launch Information

SCORM allows a LMS to define a SCO as for credit or for no-credit. It is up to the LMS what to do with the data collected from the SCO when this value is set. You have the option to make your SCO behave in a different manner if it is offered for credit or for no-credit.

function getCredit()

Parameters: none

Returns: A string containing either "credit" or "no-credit"

Example

```
/* get the credit information for this SCO */
var sCredit = getCredit();

/* see if this is a no-credit launch */
```

```
If (sCredit == "no-credit") {
        /* it is, remind the learner that he/she will not receive credit */
        …
}
```

SCORM Version	Action
SCORM 1.2	Returns the value of API.cmi.core.credit
SCORM 2004	Returns the value of API_1484_11.cmi.credit

Get the Mode Launch Information

SCORM allows a LMS to define the mode. In "normal" mode, the SCO should provide its regular instruction to the learner. The "review" mode indicates the learner is returning but the LMS will not save the record the learner's actions within the SCO. You will decide if your SCO will behave in a different manner based on the mode.

function getMode()

Parameters: none

Returns: A string containing "browse", "normal" or "review"

Example

```
/* get the mode information for this SCO */
var sMode = getMode();

/* see if this is review mode */
If (sMode == "mode") {
        /* it is, let the learner navigate to any part of the content in the SCO */
        …
}
```

SCORM Version	Action
SCORM 1.2	Returns the value of API.cmi.core.mode
SCORM 2004	Returns the value of API_1484_11.cmi.mode

Time Functions

You have the option of keeping the time of the SCOs session and taking actions based on time. The time functions provide these capabilities:

- Set the start time for the session
- Report the elapsed time for the session
- Find out how much time has been allowed for the learner's attempt on the SCO
- Find out what the SCO should do if the learner has exceeded the allotted time
- Get the total time taken so far for the learner's attempt on this SCO

Set the Start Time for the Session

You can tell the LMS the time the learner spent in the session with the SCO. If you would like to record the session time you will want to call startSessionTime() as soon as your SCO is launched.

function startSessionTime()

Parameters: none

Returns: The current Date object provided by JavaScript. You can use this value to get the current year, month, day, time and other date-related data. This function also sets a global variable named _timeSessionStart which is set to the current Date object.

Example

```
function initSCO() {
	/* tell SCORM API we have started the SCO session */
	initCommunications();

	/* remember the start time so we can record the duration of this session when
the SCO session ends */
	startSessionTime();

```

```
        …
```

SCORM Version	Action
SCORM 1.2	This function does not call and SCORM 1.2 functions
SCORM 2004	This function does not call and SCORM 2004 functions

Set the Session Time

Your SCO can set the session time. The LMS adds the times of each session together to get the total time for the learner's attempt on your SCO.

function setSessionTime()

Parameters: a JavaScript Date object that signifies the time the SCO was started

Returns: nothing

Example

```
/* record the session time */
setSessionTime(_timeSessionStart);
```

SCORM Version	Action
SCORM 1.2	Sets the value of API.cmi.core.session_time
SCORM 2004	Sets the value of API_1484_11.cmi.session_time

Get the Maximum Time Allowed for this SCO

The imsmanifest.xml file describes a SCORM course. This file can include XML data that contains the maximum amount of time allowed for the learner's attempt on your SCO. If the XML information is missing, getMaxTimeAllowed() will return "" (an empty string). The information is contained in the < adlcp:maxtimeallowed> tag for SCORM 1.2. The data is defined in the imsss:attemptAbsoluteDurationLimit attribute of the <imsss:limitConditions> tag for SCORM 2004.

Your SCO can use this data to take an action based on the elapsed time for the attempt on your SCO.

function getMaxTimeAllowed ()

Parameters: none

Returns: a string containing the maximum time in this format:

P[yY][mM][dD][T[hH][nM][s[.s]S]] where:

- y: The number of years (integer, >= 0, not restricted)
- m: The number of months (integer, >=0, not restricted)
- d: The number of days (integer, >=0, not restricted)
- h: The number of hours (integer, >=0, not restricted)
- n: The number of minutes (integer, >=0, not restricted)
- s: The number of seconds or fraction of seconds (real or integer, >=0, not restricted). If fractions of a second are used, SCORM further restricts the string to a maximum of 2 digits (e.g., 34.45 – valid, 34.45454545 – not valid).
- The character literals designators P, Y, M, D, T, H, M and S shall appear if the corresponding non-zero value is present.
- Zero-padding of the values shall be supported. Zero-padding does not change the integer value of the number being represented by a set of characters. For example, PT05H is equivalent to PT5H and PT000005H.

Here are some example elapsed times and their meaning:

- P1Y3M2DT3H indicates a period of time of 1 year, 3 months, 2 days and 3 hours
- PT3H5M indicates a period of time of 3 hours and 5 minutes

Example

```
/* get the maximum time allowed for this SCO */
var sMaxTime = getMaxTimeAllowed();

/* see if we have exceeded the time */
if (haveExceededTime(sMaxTime)) {
        /* we have, take the proper action */
        showTimeLimitExceeded();
}
```

SCORM Version	Action
SCORM 1.2	Gets the value of API.cmi.student_data.max_time_allowed
SCORM 2004	Gets the value of API_1484_11.cmi.max_time_allowed

Get the Time Limit Action for this SCO

The imsmanifest.xml file describes a SCORM course. This file can include XML data that contains the action to take when the time limit is exceeded for this SCO. If the XML information is missing, getTimeLimitAction() will return "" (an empty string). The information is contained in the < adlcp:timelimitaction> tag for SCORM 1.2. The data is defined in the tag <adlcp:timeLimitAction> for SCORM 2004.

Your SCO can use this data to decide what action to take when the time limit is exceeded.

function getTimeLimitAction()

Parameters: none

Returns: a string containing one of the following values:

- "exit,message": The learner should be forced to exit the SCO. The SCO should provide a message to the learner indicating that the maximum time allowed for the learner attempt was exceeded.
- "continue,message": The learner should be allowed to continue in the SCO. The SCO should provide a message to the learner indicating that the maximum time allowed for the learner attempt was exceeded.
- "exit,no message": The learner should be forced to exit the SCO with no message.
- "continue,no message": Although the learner has exceeded the maximum time allowed for the learnerattempt, the learner should be given no message and should not be forced to exit the SCO. This is the default value for this data model element.

Example

```
/* get the time limit action for this SCO */
var sAction = getTimeLimitAction();

/* take the appropriate action */
```

```
switch (sAction) {
        case "exit,message":
                exitWithMessage();
                break;
        case "continue,message":
                continueWithMessage();
                break;
        case "exit,no message":
                exitWithNoMessage();
                break;
        case "":
        case "continue,no message":
                continueWithNoMessage();
                break;
}
```

SCORM Version	Action
SCORM 1.2	Gets the value of API.cmi.student_data.time_limit_action
SCORM 2004	Gets the value of API_1484_11.cmi.time_limit_action

Get the Total Time

Your SCO can set the session time. The LMS adds the times of each session together to get the total time for the learner's attempt on your SCO. The LMS will set the total time to 0 seconds if your SCO does not set the session time. The total time is only set when the SCO calls the terminate function. So the actual elapsed time is the total time plus the current duration within the current session.

function getTotalTime()

Parameters: none

Returns: a string containing the maximum time in this format:

P[yY][mM][dD][T[hH][nM][s[.s]S]] where:

- y: The number of years (integer, >= 0, not restricted)
- m: The number of months (integer, >=0, not restricted)
- d: The number of days (integer, >=0, not restricted)
- h: The number of hours (integer, >=0, not restricted)
- n: The number of minutes (integer, >=0, not restricted)

- s: The number of seconds or fraction of seconds (real or integer, >=0, not restricted). If fractions of a second are used, SCORM further restricts the string to a maximum of 2 digits (e.g., 34.45 – valid, 34.45454545 – not valid).
- The character literals designators P, Y, M, D, T, H, M and S shall appear if the corresponding non-zero value is present.
- Zero-padding of the values shall be supported. Zero-padding does not change the integer value of the number being represented by a set of characters. For example, PT05H is equivalent to PT5H and PT000005H.

Here are some example elapsed times and their meaning:

- P1Y3M2DT3H indicates a period of time of 1 year, 3 months, 2 days and 3 hours
- PT3H5M indicates a period of time of 3 hours and 5 minutes

Example

```
/* get the total time of the attempt prior to the launch of this session */
var sTotalTime = getTotalTime();
```

SCORM Version	Action
SCORM 1.2	Gets the value of API.cmi.core.total_time
SCORM 2004	Gets the value of API_1484_11.cmi.total_time

State Management Functions

Your may want to have your SCO save and restore the learners actions with the SCO. Saving state lets the learner stop the SCO (end the SCO session by closing the browser or navigating to a new SCO) and then restart the SCO later as if he/she never left. Saving and restoring state can let set/get bookmarks and save/restore the learners interactions to questions in the SCO. If you want to LMS to remember the state information, you must call **learnerWillReturn(true)**. The state management functions let you:

- Set the bookmark

- Get the bookmark
- Set other state data in a data item called "suspend data"
- Get other the "suspend data"

Set the Bookmark

The bookmark is simply a string that contains information about the location within the SCO. The format and meaning of the string is only known to the SCO. The LMS simply holds the bookmark information for your SCO and returns it when requested.

function setBookmark(sBookmark)

Parameters: sBookmark – a string

Returns: nothing

Example

```
/* store the bookmark – the bookmark is an array in an index that keeps track of the
current page viewed by the learner */
/* make sure we store the index of the array as a string */
setBookmark(_nCurrentPage + "");
```

SCORM Version	Action
SCORM 1.2	Sets API.cmi.core.lesson_location (maximum 256 characters)
SCORM 2004	Returns the value of API_1484_11.cmi.location (maximum 1000 characters)

Get the Bookmark

The bookmark is simply a string that contains information about the location within the SCO. The format and meaning of the string is only known to the SCO. The LMS simply holds the bookmark information for your SCO and returns it when requested.

function getBookmark()

Parameters: none

Returns: a string

Example

```
/* not the first launch, so we can get the bookmark (the bookmark was set in a
previous session) */
var sBookmark = getBookmark();

/* go to the bookmarked page, subtract 0 to convert the bookmark from a string to a
number */
gotoPage(sBookmark-0);
```

SCORM Version	Action
SCORM 1.2	Sets API.cmi.core.lesson_location (maximum 256 characters)
SCORM 2004	Returns the value of API_1484_11.cmi.location (maximum 1000 characters)

Set the Suspend Data

The suspend data is simply a string that contains information about the state of the SCO. The format and meaning of the string is only known to the SCO. The LMS simply holds the suspend data information for your SCO and returns it when requested.

function setSuspendData(sSuspend)

Parameters: sSuspend – a string

Returns: nothing

Example

```
/* store the suspend data, we need to store information in an array plus the current
score. So, store the flattened array + a separator + the score */
setSuspendData(aResponses.join(",") + ":" + nScore);
```

SCORM Version	Action
SCORM 1.2	Sets API.cmi.suspend_data (maximum 4096 characters)
SCORM 2004	Sets API_1484_11.cmi.suspend_data (maximum 64000

	characters)

Get the Suspend Data

The suspend data is simply a string that contains information about the state of the SCO. The format and meaning of the string is only known to the SCO. The LMS simply holds the suspend data information for your SCO and returns it when requested.

function getSuspendData()

Parameters: nothing

Returns: a string

Example

```
/* get the suspend data, we previously stored a flattened array + a separator + the
score */
var sSuspend = getSuspendData(aResponses.join(",") + ":" + nScore);

/* separate the parts of the suspend data */
var aParts = sSuspend.split(":");

/* the first part contains the flattened array, recreate the array */
aResponses = aParts[0].split(",");

/* the second part contains the score, recreate the score, convert the score into a
number */
nScore = aParts[1] - 0;
```

SCORM Version	Action
SCORM 1.2	Gets the value of API.cmi.suspend_data (maximum 4096 characters)
SCORM 2004	Gets the value of API_1484_11.cmi.suspend_data (maximum 64000 characters)

Completion Functions

You will decide when your SCO is complete. You are free to select any criterion that meets your instructional requirements. For example, you could decide a SCO is complete if one of these actions is taken by the learner:

- The learner views the first page of the SCO
- All pages in the SCO have been viewed
- The learner has answered all of the questions in a test
- The learner has answered enough questions in the test to achieve a passing score
- The learner has exceeded the time allowed for the attempt on the SCO
- A simulation was completed

You can decide if the completion of the SCO should have any relationship to passing or failing the SCO. For example, you can decide the SCO should be complete if the learner passes or fails a test. You can decide that the SCO can never be completed until a learner passes a test.

The completion functions let you:

- Set the completion status
- Get the completion status
- Report the completion percentage
- Get the completion percentage (if previously set)
- Get the completion threshold

Set the Completion Status

The LMS has the ability to set the completion status by itself if the SCO does not define a completion status. So it is a good idea to mark your SCO as incomplete when the SCO is launched for the first time (when the learner is having the first session with the SCO).

The SCO can then decide which actions by the learner constitute completion of the SCO.

function setCompletionStatus(sCompletion)

Parameters: sCompletion – a string containing "completed", "incomplete" or "not attempted"

Returns: nothing

Example

```
/* the learner has completed all of the required parts of the SCO so mark the SCO
complete */
setCompletionStatus("completed");
```

SCORM Version	**Action**
SCORM 1.2	Sets API.cmi.core.lesson_status
SCORM 2004	Sets the value of API_1484_11.cmi.completion_status

Get the Completion Status

The SCO can get the completion status. This function is very useful for SCOs that contain a single Flash movie. Your Flash movie cannot tell when the HTML page containing the Flash movie is unloaded. So, your Flash movie does not have a final chance to set the completion status of your SCO. However, the HTML page does get an onunload event (and in Internet Explorer and onbeforeunload event). Your HTML page can check to the completion status to decide if it should tell the learner that he must do more to complete the SCO.

function getCompletionStatus()

Parameters: none

Returns: a string containing "completed", "incomplete", "not attempted", "unknown"

Example

```
/* see if this SCO is complete */
```

```
var sComplete = getCompletionStatus();
if (sComplete == "incomplete" || sComplete == 'unknown') {
        /* it is not, make sure the LMS retains the data so the learner can return */
        retainState(true);

        /* remind the learner that he must do more to complete the SCO */
        remindLearner();
}
```

SCORM Version	Action
SCORM 1.2	Returns API.cmi.core.lesson_status
SCORM 2004	Returns the value of API_1484_11.cmi.completion_status

Set the Completion Percentage

The SCO can tell the LMS the completion percentage. Some LMSs will show this information in a report to learners, managers and/or administrators. You will decide how your SCO calculates the completion percentage.

function setCompletionPercentage(sPercent)

Parameters: sPercent – a string containing a value between 0 and 1. 0 represents no completion. 1 represents total completion. A number in between represents a partial completion.

Returns: nothing

Example

```
/* the learner is halfway through the SCO, set the completion percentage */
setCompletionPercentage("0.5");
```

SCORM Version	Action
SCORM 1.2	There is no data item in SCORM 1.2 for the completion percentage
SCORM 2004	Sets the value of API_1484_11.cmi.progress_measure

Get the Completion Percentage

The SCO can get the completion percentage if it was previously set.

function getCompletionPercentage()

Parameters: nothing

Returns: a string containing a value between 0 and 1.

Example

```
/* get the completion percentage */
var sPercent = getCompletionPercentage();

/* show the completion percentage to the learner */
showPercentage(sPercent);
```

SCORM Version	Action
SCORM 1.2	There is no data item in SCORM 1.2 for the completion percentage
SCORM 2004	Gets the value of API_1484_11.cmi.progress_measure

Get the Completion Threshold

The SCO can get the completion threshold. The completion threshold is defined in the imsmanifest.xml file for this SCO in a SCORM 2004 course (there is not completion threshold for SCORM 1.2). The imsmanifest.xml file uses the tag <adlcp:completionThreshold> to store the completion threshold.

Your SCO can use the completion threshold to decide when the SCO is complete.

function getCompletionThreshold()

Parameters: nothing

Returns: a string containing a value between 0 and 1.

Example

```
/* get the completion threshold */
var sThreshold = getCompletionThreshold();
```

SCORM Version	Action
SCORM 1.2	There is no data item in SCORM 1.2 for the completion threshold

SCORM 2004	Gets the value of API_1484_11.cmi.completion_threshold

Pass/Fail Functions

Your SCO decides if the learner has passed or failed the attempt on the SCO. You are free to select any criterion for pass/fail that meets your instructional requirements. For example, you could decide the learner has passed the SCO when one of these actions is taken by the learner:

- The learner views the first page of the SCO
- All pages in the SCO have been viewed
- The learner has answered all of the questions in a test
- The learner has answered enough questions in the test to achieve a passing score
- A simulation was completed correctly

You can decide if the completion of the SCO should have any relationship to passing or failing the SCO. For example, you can decide the SCO should be complete if the learner passes or fails a test. You can decide that the SCO can never be completed until a learner passes a test.

The pass/fail functions let you:

- Set the pass/fail status
- Get the pass/fail status

Set the Pass/Fail Status

The LMS has the ability to set the pass/fail status by itself based on the score of the SCO. So it is a good idea to explicitly set the pass/fail when you set the SCO of the SCO.

Your SCO does not have to set the pass/fail status. However, the pass/fail status can be used to affect the sequencing decisions between SCOs in SCORM 2004 so it is a good idea to set the pass/fail status when you set the SCO's completion status.

function setPassFail(sPassFail)

Parameters: sPassFail – a string containing "passed" or "failed"

Returns: nothing

Example

```
/* the learner has completed the SCO and he/she passed, so tell the LMS the learner in
done with this attempt (learner will not return in another session) */
learnerWillReturn(false);

/* tell the LMS the learner has completed all of the work in this SCO */
setCompletionStatus("completed");

/* tell the LMS that the learner has passed */
setPassFail("passed");
```

SCORM Version	Action
SCORM 1.2	Sets API.cmi.core.lesson_status
SCORM 2004	Sets the value of API_1484_11.cmi.success _status

Get the Pass/Fail Status

Your SCO can get the pass/fail status of the SCO.

function getPassFail()

Parameters: none

Returns: a string containing "passed", "failed" or "unknown"

Example

```
/* the learner has completed the SCO and he/she passed, so tell the LMS the learner is
done with this attempt (learner will not return in another session) */
learnerWillReturn(false);

/* tell the LMS the learner has completed all of the work in this SCO */
```

```
setCompletionStatus("completed");

/* tell the LMS that the learner has passed */
setPassFail("passed");
```

SCORM Version	Action
SCORM 1.2	Gets API.cmi.core.lesson_status
SCORM 2004	Gets the value of API_1484_11.cmi.success_status

Score Functions

Your SCO determines the score for a learner based on his actions within the. You are free to select any criterion for setting the score that meets your instructional requirements. For example, you could set the score based on:

- The result of a test
- The result of multiple tests
- The actions within a simulation

You can decide if the SCO should report a score. You can decide if the score should apply the same or different weighting to questions presented to the learner within the SCO.

The score functions let you:

- Get the passing score for this SCO
- Set the score
- Get the score

Get the passing score for this SCO

The imsmanifest.xml file describes a SCORM course. This file can include XML data that contains the passing score for a SCO. If the XML information is missing, getMaxTimeAllowed() will return "1.0". The information is contained in the

<adlcp:masteryscore> tag for SCORM 1.2. The data is defined in the <imsss:minNormalizedMeasure> tag for SCORM 2004.

function getPassingScore()

Parameters: none

Returns: a string value containing a number in the range of -1.0 to 1.0

Example

```
/* the learner has completed the SCO, so tell the LMS the learner is done with this
attempt (learner will not return in another session) */
learnerWillReturn(false);

/* tell the LMS the learner has completed all of the work in this SCO */
setCompletionStatus("completed");

/* set the score and make sure to convert it to a string value */
setScore(nScore + "");

/* get the passing score and convert it to a number by subtracting 0 */
var nPassingScore = getPassingScore() - 0;

/* see if the user has passed this SCO */
If (nScore >= nPassingScore) {
        /* the learner has passed the SCO */
        setPassFail("passed");
} else {
        /* the learner has not passed */
        setPassFail("failed");
}
```

SCORM Version	Action
SCORM 1.2	Sets API.cmi.student_data.mastery_score
SCORM 2004	Sets the value of API_1484_11.cmi. scaled_passing_score

Set the Score

Your SCO can set a score for the learner's attempt on this SCO. Only one score can be reported for a SCO. So, a SCO that presents multiple tests to a learner can only report one overall score for the SCO.

function setScore()

Parameters: a string value containing a number in the range of -1.0 to 1.0

Returns: nothing

Example

```
/* the learner has completed the SCO, so tell the LMS the learner is done with this
attempt (learner will not return in another session) */
learnerWillReturn(false);

/* tell the LMS the learner has completed all of the work in this SCO */
setCompletionStatus("completed");

/* set the score and make sure to convert it to a string value */
setScore(nScore + "");

/* get the passing score and convert it to a number by subtracting 0 */
var nPassingScore = getPassingScore() - 0;

/* see if the user has passed this SCO */
If (nScore >= nPassingScore) {
        /* the learner has passed the SCO */
        setPassFail("passed");
} else {
        /* the learner has not passed */
        setPassFail("failed");
}
```

SCORM Version	Action
SCORM 1.2	Sets API.core.score.raw = 100 * the score reported by this function. A value of 0 is set if you pass a negative score. Sets API.core.score.min = "0" Sets API.core.score.max = "100"
SCORM 2004	Sets the value of API_1484_11.cmi. score.scaled

Get the Score

Your SCO can get a score if you previously set a score using setScore().

function getScore()

Parameters: none

Returns: a string value containing a number in the range of -1.0 to 1.0

Example

```
/* get the score and convert it to a number */
var nScore = getScore() - 0;
```

SCORM Version	Action
SCORM 1.2	Gets API.core.score.raw / 100
SCORM 2004	Sets the value of API_1484_11.cmi. score.scaled

Interaction Functions

Your SCO can tell the LMS about interactions. Interactions are SCORM's name for questions. You can tell the LMS quite a bit about an interaction including the ID (identifier) of the interaction, the type of interaction, the learner's response, the correct answer, whether or not the answer was correct and more.

In SCORM 1.2, you can only send information about interactions to the LMS. In SCORM 1.2, you cannot read the interactions that you previously set (in the current session or in future sessions). For example, the learner does the following:

- Enters "true" to question 1
- Enters "false" to question 2
- Changes his/her response to question 1 from "true" to "false"

SCORM 1.2 will only let you report each one of these actions as a separate interaction. So your SCO will report the interactions with code like this:

```
setInteraction(null,"Q1","true-false","true","true","correct","1.0",null,null,null);
setInteraction(null,"Q2","true-false","true","true","correct","1.0",null,null,null);
setInteraction(null,"Q1","true-false","false","true","incorrect","1.0",null,null,null);
```

SCORM 2004 lets your SCO write, read and rewrite interactions. You have the choice to report interactions in the same way as SCORM 1.2 (just write the results) or you can rewrite an interaction:

```
setInteraction("0","Q1","true-false","true","true","correct","1.0",null,null,null);
setInteraction("1","Q2","true-false","true","true","correct","1.0",null,null,null);
setInteraction("0","Q1","true-false","false","true","incorrect","1.0",null,null,null);
```

The interaction functions let you:

- Set interactions
- Get the index of an interaction based on its ID (SCORM 2004 only)

Set Interactions

SCORM defines several types of interactions:

- True-false
- Multiple Choice
- Fill-In
- Long Fill-In
- Matching
- Performance
- Sequence
- Likert
- Numeric
- Other

SCORM does not specify how your SCO presents these questions. SCORM does not specify the way the learner interacts with these questions. SCORM simply provides a way for you to report the results of the questions. For example, each of these questions would report results as a multiple choice interaction.

- A text question is shown to the learner. The learner selects the correct answer by clicking on a radio button.
- A short movie is shown to the learner. The learner is asked to navigate through the movie to select the scene introduces a specific character in the movie.
- A picture of a car engine is shown to the learner. The learner is asked to click on the fuel injector.

SCORM 2004 lets you rewrite the learner's response to a question. So, the setInteraction() function lets you set specific indexes for each interaction.

Function setInteraction(sNum, sId, sType, sResponse, sCorrect, sResult, sWeight, nLatency, sDescription, sIdObjective)

Parameters:

- **sNum** – a string representing an index. For example, "0", "5", "15". You can only pass an index when your SCO is using SCORM 2004. You can determine which version of SCORM is used by calling getCommunicationsType(). Use **null** for sNum if you do not want to set specific index numbers.
- **sId** – a string containing the ID of the interaction. Each interaction needs to have a unique ID. You can use simple ID strings such as "Q1" or "Q4". You can also use globally unique URNs as recommended described is per RFC 2141 using this format: "urn:your-unique-id:unique-id-for-the-interaction".
- **sType** - the type of the interaction, "true-false, "choice", "fill-in" "long-fill-in", "likert", "matching", "performance", "sequencing", "numeric", "other"
- **sResponse** – a string containing the response provided by the learner. The format of the response depends on the type of question. The SCORM 2004 3rd Edition Runtime Environment (SCORM_RunTimeEnv.pdf) defines the formar in section "4.2.9.2 Learner Response Data Model Element Specifics".
- **sCorrect** – a string containing the correct answer. The format of the correct answer depends on the type of question. The SCORM 2004 3rd Edition Runtime Environment (SCORM_RunTimeEnv.pdf) defines the format in section "4.2.9.1 Correct Responses Pattern Data Model Element Specifics".
- **sResult** – a string containing "correct", "incorrect", "unanticipated", "neutral" or "x.y" (a string containing a numeric value).
- **sWeight** – a string containing the weight of this question. This can be null if you do not want to report a weight for the interaction.
- **nLatency** - the time the learner took to respond to the question in milliseconds. This can be null if you do not want to report the latency for the interaction.
- **sDescription** - the description of this interaction. This can be null if you do not want to report a description for the interaction.
- **sIdObjective** – a string containing the ID of the objective associated with this interaction. This can be null if you do not want to report an objective ID for the interaction.

Returns: nothing

Example

```
function gradeQuestionTF(sResponse) {
        /* get the id of this question */
        /* the name is in a hidden field on this page */
```

```
          var sId= document.getElementsByName("qid")[0].value;

          /* get the last response to this question */
          var sLastResponse = getState(sId + "response");

          /* see if we have the same answer */
          if (sResponse == sLastResponse) {
                    /* we do, no need to do anything so quit */
                    return;
          }

          /* show the feedback */
          showFeedbackTF(sResponse);

          /* record this response */
          setState(sId + "response", sResponse);

          /* get the correct answer */
          /* it is stored in a hidden field in the question */
          var sCorrect = document.getElementsByName("correct")[0].value;

          /* grade the response */
          if (sResponse == sCorrect) {
                    var sResult = "correct";

                    /* remember the score */
                    setState(sId + "Score", "1");
          } else {
                    var sResult = "wrong";

                    /* remember the score */
                    setState(sId + "Score", "0");
          }

          /* report the interaction */
          setInteraction(null,sId,"true-
false",sResponse,sCorrect,sResult,"1",getElapsedTime(_timeStart),null,null);

          /* reset the start time */
          _dateStart = new Date();
          _timeStart = _dateStart.getTime();
}
```

SCORM Version	**Action**
SCORM 1.2	Sets: API.cmi.interaction.n.id API.cmi.interaction.n.type

	API.cmi.interaction.n.student_reponse API.cmi.interaction.n.correct_responses.0.pattern API.cmi.interaction.n.result API.cmi.interaction.n.time API.cmi.interaction.n.weighting API.cmi.interaction.n.latency API.cmi.interaction.n.objectives.0.id
SCORM 2004	Sets: API_1484_11.cmi.interaction.n.id API_1484_11.cmi.interaction.n.type API_1484_11.cmi.interaction.n.learner_reponse API_1484_11.cmi.interaction.n.correct_responses.0.pattern API_1484_11.cmi.interaction.n.result API_1484_11.cmi.interaction.n.timestamp API_1484_11.cmi.interaction.n.weighting API_1484_11.cmi.interaction.n.latency API_1484_11.cmi.interaction.n.description API_1484_11.cmi.interaction.n.objectives.0.id

Get the Index of an Interaction from Its ID

SCORM 2004 lets your SCO rewrite the data in an interaction. Your SCO will have to find the index of the interaction to rewrite it. The getInteractionIndex() returns the index of the interaction.

Function getInteractionIndex(sStart, sId)

Parameters:

- **sStart -** a string containing the starting index - use "0" if you want to start from the beginning of the list of interactions.
- **sId -** a string containing the ID of the interaction that you would like to find

Returns: a string containing the index of the interaction

Example

```
/* see if this SCO is running in SCORM 2004 */
If (getCommunicationsType( == "SCORM 2004") {
        /* it is, the learner has changed the answer for question 1, find the index for
question 1 */
        var sIndex = getInteractionIndex("0", "Q1");
        /* update the information for question 1 */
        setInteraction(sIndex, "Q1", "true-false" ,sResponse, sCorrect, sResult, "1",
getElapsedTime(_timeStart), null, null);
} else {
        /* this is SCORM 1.2, we can only add the new answer to the list of interactions
*/
        setInteraction(null, "Q1", "true-false" ,sResponse, sCorrect, sResult, "1",
getElapsedTime(_timeStart), null, null);
}
```

SCORM Version	Action
SCORM 1.2	SCORM 1.2 does not let the SCO read interactions so this function does not work with SCORM 1.2.
SCORM 2004	Gets the interaction index (n) for the ID that matches API_1484_11.cmi.interaction.n.id

Secondary Objective Functions

Your SCO can tell the LMS about the completion of objectives. There are two kinds of objectives in your SCO – primary and secondary objectives. The SCO can report only one primary objective. The SCO can report zero or more secondary objectives. The primary objective includes these elements:

Primary Objective Information	Set with this function
attempt status	learnerWillReturn()
completion status	setCompletionStatus()
completion percentage	setCompletionPercentage()
pass/fail status	setPassFail()
score	setScore()

In SCORM 2004, the primary objective can contribute to the overall completion, pass/fail status and the score of the course. The completion, pass/fail and score can be "rolled-up" by rules found in the imsmanifest.xml file. The imsmanifest.xml file can also contain sequencing rules that use the primary objective of the SCO. SCORM 1.2 does not have any roll-up or sequencing rules.

A SCO can set secondary objectives. These secondary objectives are not used directly in the roll-up or sequencing of a SCORM 2004 course. However, the secondary objectives are recorded by the LMS. The LMS can pass these secondary objectives on to other SCOs through rules set in the imsmanifest.xml file. The secondary objectives let you design a SCORM 2004 course that can provide an adaptive learning experience. For example, the first SCO presented to the learner can be a pre-test. The pre-test SCO sets secondary objectives that define how well the learner knows the subject matter going into the course. The LMS can then pass these secondary objectives to a second SCO that can provide a tutorial that only covers the material related to the objectives that the learner did pass in the pre-test.

The secondary objective functions let you:

- Set secondary objectives
- Get the index of a secondary objective based on its ID
- Get the count of secondary objectives
- Get a secondary objective score
- Get a secondary objective completion status
- Get a secondary objective completion percentage
- Get a secondary objective pass/fail status
- Get a secondary objective description

Set Secondary Objectives

SCORM does not specify how your SCO presents these questions. SCORM does not specify the way the learner interacts with these questions. SCORM simply provides a way for you to report the results of the questions. For example, each of these questions would report results as a multiple choice interaction.

- A text question is shown to the learner. The learner selects the correct answer by clicking on a radio button.
- A short movie is shown to the learner. The learner is asked to navigate through the movie to select the scene introduces a specific character in the movie.
- A picture of a car engine is shown to the learner. The learner is asked to click on the fuel injector.

SCORM 2004 lets you rewrite the learner's response to a question. So, the setInteraction() function lets you set specific indexes for each interaction.

Function setObjective(sNum, sId, sCompletion, sPercentComplete, sPassFail, sScore, sDescription)

Parameters:

- **sNum** – a string representing an index. For example, "0", "5", "15". Use **null** for sNum if you do not want to set specific index numbers.
- **sId** – a string containing the ID of the objective. Each objective needs to have a unique ID. You can use simple ID strings such as "Obj1" or "Obj4". You can also use globally unique URNs as recommended described is per RFC 2141 using this format: "urn:your-unique-id:unique-id-for-the-objective".
- **sCompletion** – a string containing the completion status of the objective - "completed", "incomplete", "not attempted", "unknown".
- **sPercentComplete** – a string containing the percent complete as a decimal value, 0 is 0% complete, 0.5 is 50% complete, 1 is 100% complete, can be null.
- **sPassFail** – a string containing the pass/fail status (progress measure) - "passed", "failed", "unknown", can be null.
- **sScore** – a string the score of the objective - a numerical value from -1 to 1, can be null.
- **sDescription** - the description of this objective. This can be null if you do not want to report a description for the interaction, can be null.

Returns: nothing

Example

```
/* the learner has completed all of the work for objective one, tell the LMS */
setObjective("0", "Obj1", "completed", "1.0", "passed", "1.0", null);
```

SCORM Version	**Action**
SCORM 1.2	Sets: API.cmi.objectives.n.id API.cmi.objectives.n.status API.cmi.objectives.n.score.min API.cmi.objectives.n.score.max API.cmi.objectives.n.score.raw
SCORM 2004	Sets: API_1484_11.cmi.objectives.n.id

	API_1484_11.cmi.objectives.n.completion_status API_1484_11.cmi.objectives.n.progress_measure API_1484_11.cmi.objectives.n.success_status API_1484_11.cmi.objectives.n.score.scaled API_1484_11.cmi.objectives.n.description

Get the Index of an Objective from Its ID

SCORM lets your SCO rewrite the data in an objective. Your SCO will have to find the index of the objective to rewrite it. The getObjectiveIndex() returns the index of the objective.

Function getObjectiveIndex(sStart, sId)

Parameters:

- **sStart -** a string containing the starting index - use "0" if you want to start from the beginning of the list of objectives.
- **sId -** a string containing the ID of the objective that you would like to find

Returns: a string containing the index of the objective

Example

```
/* find the index for question 1 */
var sIndex = getObjectiveIndex("0", "Obj1");

/* update the information for this objective */
setObjective(sIndex, "Obj1", "completed", "1.0", "passed", "1.0", null);
```

SCORM Version	Action
SCORM 1.2	Gets the interaction index (n) for the ID that matches API.cmi.interaction.n.id
SCORM 2004	Gets the interaction index (n) for the ID that matches API_1484_11.cmi.interaction.n.id

Get the Number of Secondary Objectives

Your SCO can get the total number of secondary objectives created by the SCO.

Function getObjectiveCount()

Parameters: none

Returns: a string containing the number of secondary objectives

Example

```
/* get the number of secondary objectives */
var sNum = getObjectiveCount();
```

SCORM Version	Action
SCORM 1.2	Gets API.cmi.objectives_count
SCORM 2004	Gets API_1484_11.cmi.objectives._count

Get the Score of a Secondary Objective

Your SCO can get a previously set score of a secondary objective.

Function getObjectiveScore(sIndex)

Parameters: sIndex – a string containing the index of an objective.

Returns: a string containing the score

Example

```
/* get the score of a secondary objective */
var sScore = getObjectiveScore(sIndex);
```

SCORM Version	Action
SCORM 1.2	Gets API.cmi.objectives.n.score.raw / 100
SCORM 2004	Gets API_1484_11.cmi.objectives.n.score.scaled

Get the Completion Status of a Secondary Objective

Your SCO can get a previously set completion status of a secondary objective.

Function getObjectiveCompletionStatus (sIndex)

Parameters: sIndex – a string containing the index of an objective.

Returns: a string containing the completion status

Example

```
/* get the completion status of a secondary objective */
var sStatus = getObjectiveCompletionStatus (sIndex);
```

SCORM Version	Action
SCORM 1.2	Gets API.cmi.objectives.n.status
SCORM 2004	Gets API_1484_11.cmi.objectives.n.completion_status

Get the Completion Percentage of a Secondary Objective

Your SCO can get a previously set completion percentage of a secondary objective.

Function getObjectiveCompletionPercentage(sIndex)

Parameters: sIndex – a string containing the index of an objective.

Returns: a string containing the completion percentage

Example

```
/* get the completion percentage of a secondary objective */
var sPercentComplete = getObjectiveCompletionPercentage(sIndex);
```

SCORM Version	Action
SCORM 1.2	SCORM 1.2 does not define this type of value for an objective. The function returns "" for SCORM 1.2
SCORM 2004	Gets API_1484_11.cmi.objectives.n.progress_measure

Get the Pass/Fail Status of a Secondary Objective

Your SCO can get a previously set pass/fail status of a secondary objective.

Function getObjectivePassFail(sIndex)

Parameters: sIndex – a string containing the index of an objective.

Returns: a string containing the pass/fail status

Example

```
/* get the completion percentage of a secondary objective */
var sPassFail = getObjectivePassFail(sIndex);
```

SCORM Version	Action
SCORM 1.2	SCORM 1.2 does not define this type of value for an objective. The function returns "" for SCORM 1.2
SCORM 2004	Gets API_1484_11.cmi.objectives.n.success_status

Get the Description of a Secondary Objective

Your SCO can get a previously set description of a secondary objective.

Function getObjectiveDescription(sIndex)

Parameters: sIndex – a string containing the index of an objective.

Returns: a string containing the description

Example

```
/* get the description of a secondary objective */
var sDescription = getObjectiveDescription(sIndex);
```

SCORM Version	Action
SCORM 1.2	SCORM 1.2 does not define this type of value for an objective. The function returns "" for SCORM 1.2
SCORM 2004	Gets API_1484_11.cmi.objectives.n.description

Type of Communications Functions

Theses functions let your SCO know the specific type of communications available to your SCO. You SCO can be launched from:

- A LMS that supports SCORM 2004
- A LMS that support SCORM 1.2
- As a web page from a folder on a local computer

When you are creating your SCO you will probably need to test it frequently by launching the SCO as a web page from a folder on a local computer. The toolkit functions will continue to work when you launch your SCO from the local computer however your SCO will always appear to be in its first session. None of the data (for example, the bookmark) that you set will be kept (there is no LMS available to store your data).

Determine If the SCO Can Communicate With the LMS

You may want to call diagnostic code if your SCO cannot communicate with the LMS.

function canCommunicateWithLMS()

Parameters: none

Returns: true if the SCO can communicate with the LMS, else false

Example

```
/* see if this SCO can communicate with the LMS */
if (canCommunicateWithLMS() == false) {
        /* no communication available, put up an alert */
        alert("The SCO cannot communicate with the LMS");
}
```

SCORM Version	Action
SCORM 1.2	Returns true if the SCORM 1.2 object API can be located
SCORM 2004	Returns true if the SCORM 2004 object API_1484_11 can be located

Get the SCORM Version

The toolkit does it best to perform the same functions with both SCORM 1.2 and SCORM 2004. However, there are some differences between the two versions of SCORM. So, you may want your SCO to get the SCORM version so you can provide code to handle these differences.

function getCommunicationsType()

Parameters: none

Returns: a string containing "SCORM 2004", "SCORM 1.2 or "none" (launched as a web page from the local computer)

Example

```
/* see if this SCO is running in SCORM 2004 */
If (getCommunicationsType( == "SCORM 2004") {
        /* it is, the learner has changed the answer for question 1, find the index for
question 1 */
        var sIndex = getInteractionIndex("0", "Q1");

        /* update the information for question 1 */
        setInteraction(sIndex, "Q1", "true-false" ,sResponse, sCorrect, sResult, "1",
getElapsedTime(_timeStart), null, null);
} else {
        /* this is SCORM 1.2, we can only add the new answer to the list of interactions
*/
        setInteraction(null, "Q1", "true-false" ,sResponse, sCorrect, sResult, "1",
getElapsedTime(_timeStart), null, null);
}
```

SCORM Version	Action
SCORM 1.2	Returns " SCORM 1.2" if the SCORM 1.2 object API can be located
SCORM 2004	Returns "SCORM 2004" if the SCORM 2004 object API_1484_11 can be located

Lower Level Functions

The lower level functions let you directly call the SCORM 1.2 and SCORM 2004 Runtime API provided by the LMS. You can call these lower level functions if you need a specific SCORM feature not provided by the higher-level functions.

Toolkit Function	SCORM 2004 Function	SCORM 1.2 Function
scormInitialize()	Initialize("")	LMSInitialize("")
scormTerminate()	Terminate("")	LMSFinish("")
scormGetValue(sName)	GetValue(sName)	LMSGetValue(sName)
scormSetValue(sName, sValue)	SetValue(sName, sValue)	LMSSetValue(sName, sValue)
scormGetLastError()	GetLastError()	LMSGetLastError()
scormGetErrorString()	GetErrorString()	LMSGetErrorString()
scormGetDiagnostic()	GetDiagnostic()	LMSGetDiagnostic()

Sample SCOs

The toolkit comes with five sample SCOs that demonstrate different ways to use the toolkit.

- Sample 1 – This SCO shows you how to create a multi-page SCO without the use of a HTML frameset. The SCO presents several information pages to the learner using individual HTML pages. The SCO is complete when the learner reaches the last HTML page.
- Sample 2 – This SCO is similar Sample 1. However, the launch page (described below) is implemented differently than in sample 1.
- Sample 3 – This SCO is similar in design to samples 1 and 2. This SCO presents a simple test. The first page of the SCO provides an introduction to the test. The next two pages present questions. The final page shows a summary of the test. The SCO is complete when the learner achieves a passing score.
- Sample 4 – This SCO is implemented in a frameset that contains two frames. The frameset is available throughout the SCORM session so it provides a convenient place to maintain the state information for the SCO. However, a frameset makes it that much more difficult for visually impaired learners to use a screen reader to understand the content of the course. SCOs contain state information so it is usually easier to create a SCO with a frameset. The top frame of the frameset presents the content (several HTML pages). The bottom frame contains the navigation buttons used to navigate within the SCO.
- Sample 5 – This SCO is implemented in Flash. The SCO contains a HTML page which contains a single Flash movie (SWF file). All of the content of the SCO is contained within the Flash movie. The movie uses ActionScript to get/set the bookmark and to set the completion status of the SCO.

The samples are designed to show you different ways to use the functions contained in this toolkit. The samples are sparse so you can easily see the code needed to communicate with the LMS using SCORM. The samples do not contain rich media and other interactivity that will be unique to your real-life courses. Once you understand how the samples implement SCORM you can add the real-life media and interactivity needed to support the instructional-design of your course.

You can also study the samples to see how SCORM support could be added to existing courses.

Sample 1 – a SCO Launched without a Frameset

This SCO shows you how to create a multi-page SCO without the use of a HTML frameset. Many course developers find it easier to create SCOs with a HTML frameset. Sample 4 shows you how to create a SCO using a frameset.

The SCO in sample 1 presents several information screens to the learner using individual HTML pages. The attempt on the SCO is complete when the learner reaches the last HTML page (**summary.htm**). Sample 1 contains these files:

- Launch page – the LMS loads **sample1\launch.htm** into the browser to launch the SCO.
- Content pages – the content of the SCO is in the **sample1** folder. The content consists of **page1.htm**, **page2.htm**, **page3.htm** and **summary.htm**.
- Style sheet – all of the content pages use a common CSS style sheet named **style.css**.
- JavaScript files – the SCO's launch page and content pages use three JavaScript files.
    - **sco_api.js** – this file contains all of the toolkit's functions.
    - **sco_noframes.js** – this file contains JavaScript functions that are useful to create SCOs that do not use a frameset to launch the SCO. **sco_noframes.js** is used in samples 1, 2 and 3.
    - **sco_complete.js** – this file contains JavaScript functions called when the pages in the SCO are unloaded

The SCO's Launch File – launch.htm

A SCO must contain at least one HTML page. Sample 1 contains several HTML pages. The first page launched by the LMS for this SCO is **launch.htm**. **launch.htm** decides if the SCO should show the learner the beginning of the SCO (page1.htm) or pick up from the bookmarked page recorded by the SCO when it was launched in a previous session.

Launch.htm calls the function **initSCO()** which initializes the SCO (the function **initSCO()** is defined in **sco_noframes.js**). **initSCO()** takes a single parameter. If you pass a **null** as the parameter, the function returns as soon as the SCO initialization is complete. If you pass a file name as a parameter, the function will do one of two things:

1. The file name passed to i**nitSCO()** will be loaded into the browser if this is the first time the learner has launched this SCO.

2. The bookmarked page will be loaded if this is **not** the first time the learner has launched the SCO.

The launch page for the first sample SCO (**sample1\launch.htm**) automatically launches either the first page of the SCO or the bookmarked page. The launch page does this by using the **onload** attribute for the <body> tag.

```
<body onload="apiInitSCO('page1.htm')">
```

The Content Pages in the SCO

The SCO in sample 1 is simply a collection of HTML pages that present the content of the SCO. Sample 1 shows the pages in a simple linear order. Your SCO could use a linear order, let the learner select the order from a menu or use some other method to provide navigation from page to page. The pages in your SCO can also provide any type of information or interaction.

Each page in this sample work like this:

1. Each page includes the JavaScript code from the toolkit (**sco_api.js, sco_noframes.js and sco_complete.js**).

2. The pages in the SCO link to other pages in the SCO in a specific way.

3. The pages call a function named **unloadPage()** when they unload. The summary.htm page calls the **unloadPage()** function with a parameter to set the completion status of the SCO.

Including the JavaScript Files

Each page in the SCO includes **sco_api.js, sco_noframes.js** and **sco_complete.js**. The file **sco_api.js** contains the toolkits functions. The file **sco_noframes.js** contains JavaScript functions that make it easy to create a SCO without a frameset. The functions in **sco_complete.js** are called when a page is unloaded in the SCO. The **sco_complete.js** file contains functions can be customized for each SCO (more details below). The links to the JavaScript code are in the <head> section of the HTML page:

```
<script language="javascript" type="text/javascript" src="../sco_api.js"></script>
<script language="javascript" type="text/javascript" src="../sco_noframes.js"></script>
<script language="javascript" type="text/javascript" src="sco_complete.js"></script>
```

Links in the Content Pages

The content pages in this sample provide links between the pages. For example, Page 2 (**sample1\page2.htm**) has a link back to the page 1 and a link forward to the page 3. The links between content pages have the same format:

```
<a href="page2.htm" onclick="apiLink(this.href); return false;"
onkeypress="apiLink(this.href); return false;" alt="Link to page 2">Go to page 2</a>
```

- **href="page2.htm"** – the filename of the page to link to
- **onclick="gotoPage(this.href); return false;"** – this JavaScript function is called when the learner clicks on the link. This function loads the file specified in the **href** attribute.
- **onkeypress="gotoPage(this.href); return false;"** - this JavaScript function is called when the learner selects the link with a key press. This function loads the file specified in the **href** attribute.

The **gotoPage()** function (found **in sco_noframes.js**) keeps track of the bookmark for the SCO. The bookmark records the learner's current location with a SCO. When the learner restarts the SCO (launches a new session) the bookmark information is retrieved and the learner is returned to the previous location within the SCO.

Unloading a Content Page

When a content page in the SCO is unloaded, it calls an event named **onunload** (Internet Explorer also calls an event named **onbeforeunload**). These events are attributes of the <body> tag. Each page in the SCO calls the **unloadPage()** function to handle these events.

```
<body onbeforeunload="unloadPage()" onunload="unloadPage()">
```

The **unloadPage()** function automatically keeps track of state information between pages.

The **unloadPage()** function calls two functions in the **sco_complete.js** file. You could modify these functions for every SCO in your course. The functions are:

function stateCheck(sCompletion) – called every time your page is about to unload. You can use this function to remember state information for your page. The parameter **sCompletion** is the data passed to **unloadPage()**

function completionCheck(sCompletion) – called only when the SCO session will be terminated You can use this function to set the SCO completion status. The parameter **sCompletion** is the data passed to **unloadPage()**

Completing the Attempt on the SCO

The final page in the SCO is **summary.htm**. **summary.htm** calls **unloadPage('complete')** when it is unloaded. This function end ups up calling the **completionCheck()** function in **sco_completion.js** (shown below). This function decides if the SCO session should be marked as completed and the SCO should tell the LMS that the attempt on this SCO is complete (the learner will not return in another session).

```
function completionCheck(sCompletion) {
        /* see if the page has told us to mark the SCO as complete */
        if (sCompletion == "complete") {
                /* it has, mark the SCO complete */
                setCompletionStatus("completed");

                /* view this as a SCO as passed */
                setPassFail("passed");

                /* the learner will not return */
                learnerWillReturn(false);
        } else {
                /* in has not, set the SCO to incomplete */
                setCompletionStatus("incomplete");

                /* the learner will  return */
                learnerWillReturn(true);
        }
}
```

Sample 2 – a Variation to Sample 1

Sample 2 is identical to sample 1 except for the launch page (**sample2\launch.htm**). The launch page for the second sample SCO (**sample2\launch.htm**) initializes the SCO by calling **initSCO()** with the parameter set to null (see code sample below). The launch page then displays a message to the learner. The learner gets one message when he or she first launches the SCO. The learner gets another message if he or she has launched the SCO one or more times.

```
<h1>Sample SCO 2 Launch Page</h1>
<p>This is the launch page of the SCO. This portion of the page will be displayed every
time the learner launches this SCO.</p>
<script language="javascript" type="text/javascript">
        /* Initialize the SCO session */
        initSCO(null);

        /* initialize a variable to write dynamic information to this page */
        var sData = '<p>This portion of the page is dynamically generated. ';

        /* see if this is the learner's first time launching this SCO */
        if (isFirstLaunch()) {
                /* it is, write welcome information and provide a link to the first page
of content */
                sData += 'This is the first time you have launched the SCO. ';
                sData += 'Click the link below to go to the first page.</p>';
                sData += '<div class="nav">';
                sData += '<a id="nextlink" href="page1.htm"
onclick="gotoPage(this.href); return false;" onkeypress="gotoPage(this.href); return
false;" title="Link to page 1">Go to page 1</a>';
                sData += '</div>';
        } else {
                /* get the bookmark */
                var sBookmark = getBookmark();

                /* welcome the user and provide a link to it */
                sData += 'You have launched this SCO before. ';
                sData += 'Click the link below to return to the bookmarked page.</p>';
                sData += '<div class="nav">';
                sData += '<a id="nextlink" href="' + sBookmark + '"
onclick="gotoPage(this.href); return false;" onkeypress="gotoPage(this.href); return
false;" title="Link to the bookmarked page">Go to the bookmarked page</a>';
                sData += '</div>';
        }

        /* write out the HTML */
        document.write(sData);
</script>
```

You can decide if you prefer the method in sample 1 or sample 2 to launch your SCOs created without a frameset. Both methods conform to the SCORM standard. The second method might be better suited to learners with a visual disability since it allows the learner to control the navigation to the first page of content in the SCO.

Sample 3 – a Test in a SCO Launched without a Frameset

Sample 3 is very similar to sample 1. Both samples create a SCO without a frameset. Both SCOs show a linear collection of HTML pages. The SCO in sample 1 shows a sample test. Sample 3 shows you how to use the toolkit functions to save and restore information collected by the SCO. This information is used to collect the learner's responses to the test questions and to keep track of the score for each question. The SCO uses this information to calculate an overall score for the SCO.

The Question Pages

Sample 3 has two HTML pages that contain true-false pages. The question pages are **sample3\q1.htm** and **sample3\q2.htm**. The pages present a question to the learner and then let the learner select a radio button to indicate a response of true or false. The radio buttons call a the function **gradeQuestionTF()** to grade the response, show the appropriate feedback, tell the LMS about this interaction and save information about the response and score in the suspend data for this SCO.

You may want to implement questions in your SCOs. It is helpful to trace through the logic in this sample to see how a question can save and restore plus contribute to the overall score of a SCO.

The Summary Page

Sample 3 has a summary page (**summary.htm**). The summary page gets the information set by the question pages. The summary pages checks to see scores given to each question in the SCO. The summary page then displays a dynamic message to tell the learner if he has passed or failed the SCO. You must launch the summary page from a LMS (or a tool such as SCORM Visualizer (http://www.e-learningconsulting.com/products/scorm-visualizer.html) to see the summary page work correctly. The summary page depends on getting the suspend data for the SCO. The suspend data is not available when the SCO is launched from a folder on your computer.

Sample 4 – a SCO Launched with a Frameset

This SCO shows you how to create a multi-page SCO with the use of a HTML frameset. Many course developers find it easier to create SCOs with a HTML frameset. Samples 1, 2 and 3 show you how to create a SCO using a frameset.

The SCO in sample 4 presents several information screens to the learner using individual HTML pages. The attempt on the SCO is complete when the learner reaches the last HTML page (**summary.htm**). Sample 4 contains these files:

- Launch page – the LMS loads **sample4\launch.htm** into the browser to launch the SCO. This page is a frameset that contains 2 frames. **blank.htm** is initially loaded into the top frame. **navigation.htm** is loaded into the bottom frame.
- Content pages – the content of the SCO is in the **sample4** folder. The content consists of **page1.htm**, **page2.htm**, **page3.htm** and **summary.htm**.
- Style sheet – all of the content pages use a common CSS style sheet named **style.css**.
- JavaScript files – the SCO's launch page uses two JavaScript files.
    - **sco_api.js** – this file contains all of the toolkit's functions.
    - **sco_frameset.js** – this file contains JavaScript functions that are useful to create SCOs that use a frameset to launch the SCO.

The SCO's Launch File – launch.htm

A SCO must contain at least one HTML page. Sample 4 contains several HTML pages. The first page launched by the LMS for this SCO is **launch.htm**. **launch.htm** decides if the SCO should show the learner the beginning of the SCO (page1.htm) or pick up from the bookmarked page recorded by the SCO when it was launched in a previous session.

Launch.htm calls the function **initSCO()** when it is loaded. This function initializes the SCO session (the function **initSCO()** is defined in **sco_frameset.js**) and then does one of two things:

1. Loads the file name in **_aPages[0]** into the browser if this is the first time the learner has launched this SCO. The SCO completion status is set to "incomplete" if this is the first launch of the SCO by the learner.

2. The bookmarked page will be loaded if this is **not** the first time the learner has launched the SCO.

The **initSCO()** function also records the start time for the session. The launch page calls **termSCO()** when it is unloaded. **termSCO()** records the session time and terminates the session. The launch page calls initSCO() and termSCO() by attaching those function calls to the onload, onbeforeunload and onunload attributes in the <frameset> tag.

```
<frameset rows="*,40" frameborder="no" border="0" framespacing="0" onload="initSCO()"
onbeforeunload="termSCO()" onunload="termSCO()">
```

The Content Pages in the SCO

The SCO in sample 4 is simply a collection of HTML pages that present the content of the SCO. Sample 4 shows the pages in a simple linear order. Your SCO could use a linear order, let the learner select the order from a menu or use some other method to provide navigation from page to page. The pages in your SCO can also provide any type of information or interaction.

Each page in this sample work like this:

1. Each page has the option to call the functions located in the frameset defined in **launch.htm**. For example, the summary page (**summary.htm**) marks the SCO as complete by calling **parent.setCompletionStatus("completed")**.

2. The pages in the SCO link to other pages in the SCO with simple anchor (<a>) links. The pages are automatically loaded into the upper frame of the frameset.

Including the JavaScript Files

Only the launch page (**launch.htm**) loads the JavaScript files for the toolkit. The content pages call these functions from the frameset. The links to the JavaScript code are in the <head> section of the **launch.htm** page:

```
<script language="javascript" type="text/javascript" src="../sco_api.js"></script>
<script language="javascript" type="text/javascript" src="sco_frameset.js"></script>
```

Navigation within the SCO

The bottom frame of the frameset loads a file named **navigation.htm**. This file defines two buttons to move forward and back within the SCO. You could use this type of navigation within your SCO. You could also provide a frame that shows a table of contents. You could put the navigation buttons within each page. You could create a course that has non-linear navigation. The complexity and the function of the navigation can be driven by the instructional design of your SCO.

The navigation in this sample is rather simple. We will trace the navigations steps. **navigation.htm** defines functions for the back and previous buttons to call

```
<a id="prevlink" href="#" onclick="prevPage(); return false;" onkeypress="prevPage();
return false;" title="Go to the previous page">&lt;&lt;Back</a>

<a id="nextlink" href="#" onclick="nextPage(); return false;" onkeypress="nextPage();
return false;" title="Go to the next page">Next&gt;&gt;</a>
```

The **nextPage()** function (defined in **sco_frameset.js**) is called when the learner clicks on the next button. The **nextPage()** function calls the **gotoPage()** function to load the next content page. The updates a variable named **_nCurrentPage** which is an index into an array of page names called **_Pages**. The **gotoPage()** function sets the bookmark (so the SCO can restart from the booked mage in the next session) and then loads the next file name from the array into the content frame of the frameset.

```
function gotoPage(nIndex) {
	/* update the current page global variable */
	_nCurrentPage = nIndex;

	/* see if this is the first page */
	if (nIndex == 0) {
		/* it is, make the previous button invisible so the learner cannot select
it */
		navigationFrame.document.getElementById("prevlink").style.visibility =
"hidden";
	} else {
		/* not the first page, make sure the previous butotn is visible */
		navigationFrame.document.getElementById("prevlink").style.visibility =
"visible";
	}

	/* see if this is the last page */
	if (nIndex == _aPages.length - 1) {
		/* it is the last page, make the next button invisible so the learner
cannot select it */
		navigationFrame.document.getElementById("nextlink").style.visibility =
"hidden";
	} else {
		/* not the last page, make sure the next butotn is visible */
		navigationFrame.document.getElementById("nextlink").style.visibility =
"visible";
	}

	/* set the bookmark so we can return to this page when the SCO is relaunched
*/
	/* make sure we pass a string to the setBookmark functions since all data in
SCORM is passed as a string */
	setBookmark(nIndex+"");

	/* load the content page into the content frame */
	contentFrame.location.replace(_aPages[nIndex]);
}
```

Completing the Attempt on the SCO

The final page in the SCO is **summary.htm**. **summary.htm** marks the SCO as complete, passes and tells the LMS that the attempt on this SCO is complete (the learner will not return in another session).

```
function scoIsDone() {
        if (self != parent) {
                parent.setCompletionStatus("completed");
                parent.setPassFail("passed");
                parent.learnerWillReturn(false);
        }
}
</script>
</head>
<body onload="scoIsDone()">
```

Sample 5 – A SCO Created With Flash

This SCO shows you how to create a SCO with a Flash movie (a SWF file). The SCO in sample 5 presents several information screens to the learner using frames in a single Flash movie. The attempt on the SCO is complete when the learner reaches the last set of frames. Sample 5 contains these files:

- Launch page – the LMS loads **sample5\launch.htm** into the browser to launch the SCO. This page loads a single Flash movie called **sample5\sample5.swf**.

- Content – the content of the SCO is a collection of frames in **sample5\sample5.swf**. The SWF file is created with **sample5\sample5.fla** and **sample5\sample5.as**.

- JavaScript files – the SCO's launch page uses three JavaScript files.

  - **sco_api.js** – this file contains all of the toolkit's functions.

  - **sco_ending.js** – this file contains a JavaScript function that iscalled when the launch page is unloaded. A Flash movie cannot reliably detect when it is unloaded so this function can be used to perform the final calls to the toolkits functions.

  - **ufo.js** – this file is used by **sample5\launch.htm** to load the Flash movie. Recent changes to Internet Explorer require a Flash movie to be "activated" (clicked on) before input is allowed. The functions in **ufo.js** use JavaScript to load the Flash movie so it does not have to be activated. **ufo.js** contains code written by Bobby Vandersluis http://www.bobbyvandersluis.com/ufo/.

The SCO's Launch File – launch.htm

A SCO must contain at least one HTML page. So we must have a launch page even though we are created a SCO using Flash.

Launch.htm calls the function **initSCO()** when it is loaded. This function initializes the SCO session (the function **initSCO()** is defined in **launch.htm**). The **initSCO()** function initializes the SCORM session and also records the start time for the session.

The launch page calls **termSCO()** when it is unloaded. **termSCO()** records the session time, calls a function called **scoIsEnding()** in **sco_ending.js** and terminates the session. The launch page calls initSCO() and termSCO() by attaching those function calls to the onload, onbeforeunload and onunload attributes in the <frameset> tag.

```
/* load the Flash movie using UFO code */
var FO = { movie:"sample5.swf", width:"100%", height:"100%", majorversion:"8",
build:"0", id:"ufoCom", name:"ufoCom", swliveconnect:"true",
allowscriptaccess:"always", setcontainercss:"true" };
UFO.create(FO, "ufo");

/* onload function */
function initSCO() {
        /* start the SCORM session */
        initCommunications();

        /* start the session timer */
        startSessionTime();

        /* add the unload function - we need to do this here because UFO also has an
unload handler */
        addOnunloadEvent(termSCO);
}

/* unload function */
_bUnload = false;
function termSCO() {
        /* see if we have already done this */
        if (!_bUnload) {
                /* we have not, remember */
                _bUnload = true;

                /* set the session time */
                setSessionTime(_timeSessionStart);
```

```
                        /* see if this SCO is complete */
                        var sComplete = getCompletionStatus();
                        if (sComplete == "incomplete" || sComplete == 'unknown') {
                                /* it is not, make sure the LMS retains the data so the learner
can return */
                                learnerWillReturn(true);
                        }

                        /* allow the SCO author to make one last SCORM call before the session
is complete */
                        scoIsEnding();

                        /* terminate the session */
                        termCommunications();
            }
}
```

sample5\sample5.swf loads an ActionScript file called **sample5\sample5.as** to perform most of its functions. Maintaining your ActionScript code in a separate file makes it easy to create SCOs that can be reskinned easily without changing the functionality of the SCO by mistake. **sample5\sample5.swf** has no visual element on the first frame of the movie. Having a visually-blank first frame is typical when you create a SCO using Flash. The first frame has ActionScript code (found in **sample5\sample5.as**) that checks to see if the learner is launching this SCO for the first time. If he/she is launching the SCO for the first time, the code calls goToAndPlay(_movieStartFrame) to start playing the first part of the SCO. If the learner is returning, the code gets the bookmark (a frame number) and starts playing the movie from the bookmark. Each section within the movie updates the bookmark. The movie provides a way to navigate to the different sections within the movie.

The Content of the SCO

The SCO in sample 5 is a collection of Flash frames in **sample5.swf**. The sample flash file defines 3 ranges of frames that are labeled "first", "second" and "third". You could design your Flash movie in any way. For example, you could have a simple movie that plays from start to end. You could create a Flash movie that loads many other Flash movies. There is no limit to the functionality that you can add to your Flash movie.

The Flash movie can make calls to the toolkit's JavaScript functions. These function calls are made from ActionScript to JavaScript using a function in Flash 8 called

External Interface. For example, here is how the Flash movie uses External Interface to tell the LMS that the SCO's completion status is incomplete:

```
ExternalInterface.call("setCompletionStatus","incomplete");
```

Including the JavaScript Files

Only the launch page (**launch.htm**) loads the JavaScript files for the toolkit. The links to the JavaScript code are in the <head> section of the **launch.htm** page:

```
<script language="javascript" type="text/javascript" src="../sco_api.js"></script>
<script language="javascript" type="text/javascript" src="sco_ending.js"></script>
```

The Flash movie can make calls to the toolkit's JavaScript functions. These function calls are made from ActionScript to JavaScript using a function in Flash 8 called External Interface. For example, here is how the Flash movie uses External Interface to tell the LMS that the SCO's completion status is incomplete:

```
ExternalInterface.call("setCompletionStatus","incomplete");
```

Navigation within the SCO

The movie provides buttons to navigate to the different sections within the movie.

```
buttonListener1 = new Object();
…

// these functions cause each button to go to the correct section of the movie
buttonListener1.click = function (){ gotoAndStop("first");}
first_btn.addEventListener("click", buttonListener1);
…
```

You could define any type of navigation within your Flash movie for your SCO. You will just want to provide a way to update the bookmark for each navigation request.

Completing the Attempt on the SCO

The final section in the Flash movie marks the SCO as complete, passes and tells the LMS that the attempt on this SCO is complete (the learner will not return in another session).

```
/* use the ExternalInterface object for all SCORM communications */
import flash.external.*;

// learner has done everything needed to finish this SCO so mark the SCO as complete
ExternalInterface.call("setCompletionStatus","completed");

// learner has performed done a great job in the SCO so mark the SCO as passed
ExternalInterface.call("setPassFail","passed");

// tell the LMS that the learner does not plat to return to the SCO
ExternalInterface.call("learnerWillRelaunch",false);
```

Creating Your Own SCO

The samples provide a good framework to create your own SCOs. Take a look at the sample that best fits your development needs:

- **Create a SCO launched with a HTML Frameset** – launching a SCO with a frameset provides the easiest way to manage the state information throughout the pages in your SCO. The frameset also provide a place to centralize the code needed to implements your SCOs instructional strategy. Sample 4 demonstrates how to create SCO using a HTML frameset. Your HTML pages can contain any type of content supported by the browser including Flash movies.

- **Create a SCO WITHOUT using a HTML Frameset** – some organizations do not want to use framesets to create a SCO because framesets makes it more difficult for visually impaired learners to use the SCO. One thing to keep in mind. Most LMS systems launch your SCO inside a frameset. The LMS's frameset contains the SCORM Runtime API and a frame used to present the navigation provided by the LMS. Samples 1, 2 and 3 demonstrate how to create a SCO without using a HTML frameset. These samples use JavaScript functions in **no_frames.js** which greatly reduce the work to create a SCO without a launching frameset. Your HTML pages can contain any type of content supported by the browser including Flash movies.

- **Create a SCO with Flash** – You may want to create all of the content of your SCO in Flash. Flash provides a great user experience (of course the learner will need to have the Flash plug-in to use a SCO created with Flash). Flash provides excellent control over animation, audio and video. You can use Flash to create sophisticated interactions. Sample 5 demonstrates how to create a SCO using Flash.

Testing the Functionality of Your SCO

There are three functionality areas that you should test for your SCO:

1. Does the SCO use the SCORM Runtime API correctly
2. Does the SCO report the correct information to support the instructional design of the SCO and the course
3. Does the SCO correctly save and restore state so it can be launched in multiple sessions

Test the Use of the SCORM Runtime API

ADL provides a SCORM conformance test. There are two versions of the conformance test – one for SCORM 2004 and one for SCORM 1.2.

- SCORM 2004 - SCORM 2004 3rd Edition Conformance Test Suite Version 1.0. http://www.adlnet.gov/downloads/306.cfm
- SCORM 1.2 - SCORM Version 1.2 Conformance Test Suite Version 1.2.7. http://www.adlnet.gov/downloads/194.cfm

The conformance test will test the minimum conformance level of your SCO with the SCORM specification. The conformance test will indicate if a SCO:

1. Finds the SCORM Runtime Adapter
2. Calls the initialize and terminate functions at the right time
3. Has the proper syntax for making other SCORM function calls

The conformance test will not tell you if your SCORM reports data items essential to your instructional design (such as the completion status and a score). The conformance test will also not tell you if your SCO manages state correctly in multiple sessions (the conformance test only launches your SCO once).

Test the Information Reported to Support Your Instructional Design

You will want to make sure your SCO reports on the data required for the instructional design of the SCO and the course containing your SCO. Generally, you will want to create SCOs that report some or all of these items:

Information to report	Set with this function
attempt status	learnerWillReturn()
completion status	setCompletionStatus()
completion percentage	setCompletionPercentage()
pass/fail status	setPassFail()
score	setScore()

The information you report from your SCO can be used within a SCORM 2004 course to affect the sequencing of the course. The information reported by the SCO can also be rolled-up into an overall result for the course. For example, the overall score of the course can be the average of the scores reported by three SCOs.

The ADL conformance test will show you the information reported by your SCO in its initial launch. LMSs will show you some or all of the information reported by a SCO. SCORM Visualizer (http://www.e-learningconsulting.com/software.html) will show you all of the information reported by your SCO at the exact moment it is reported.

Test the Use or State Data to Support Multiple Launches of Your SCO

You may want to create a SCO that restore the bookmark and other state information when the SCO is relaunched by a learner. SCORM Visualizer (http://www.e-learningconsulting.com/software.html) will show you all of the state information and will let you relaunch the SCO so you can see if the state information is used correctly. LMSs will also let you relaunch a SCO but most do not let you view the state information between launches.

Publishing Your SCORM Course

A SCORM course is a collection of one or more SCOs. You publish a course by creating a SCORM manifest. The SCORM manifest is a file named imsmanifest.xml. You then place the manifest file and the SCOs into a zip file. The zip file is called a Package Interchange Format (PIF) file. Most people refer to this zip file as a SCORM package.

The manifest file describes:

- The metadata for the course. This can include the title and description of the course.
- The organization of the SCOs. This can include the title of each SCO. The LMS uses the organization and title of the SCOs to construct a table of contents of the course.
- The names of the launch files for each SCO. This tells the LMS which file to load into the browser to launch each SCO.
- A list of all of the files used by each SCO.
- Sequencing and roll-up rules – this is only available in a SCORM 2004 course.

Create a SCORM Manifest

A SCORM manifest is an XML file that describes the contents of a course. There are free tools that let you create a SCORM manifest:

- SCORM 2004 – The RELOAD Editor - http://www.reload.ac.uk/editor.html
- SCORM 1.2
  - The RELOAD Editor - http://www.reload.ac.uk/editor.html

Create a SCORM Package

A SCORM Package is used to publish a SCORM course. A SCORM package is a zip file that contains:

- imsmanifest.xml and the XSD files required to support the manifest. The imsmanifest.xml must be in the root of the zip file.
- All of files used by the SCOs

Windows XP and Windows Vista have a built-in ability to create zip files. You can also use tools such as winzip to create a zip file.

Sample Courses

The toolkit contains two sample courses. Both of the samples contain the same five sample SCOs. One sample course uses a SCORM 2004-format package. The other sample course uses a SCORM 1.2-format package.