Unity-SCORM Integration Kit

Developers Guide

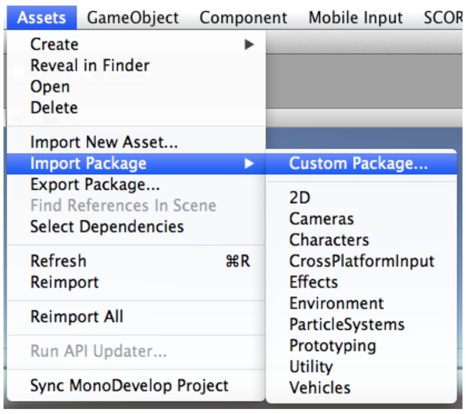
This guide explores the functionality of the Unity-SCORM Integration Kit. It is designed for Unity3D developers who are looking to integrate their projects with Learning Management Systems via SCORM.

Getting Started in Unity3D

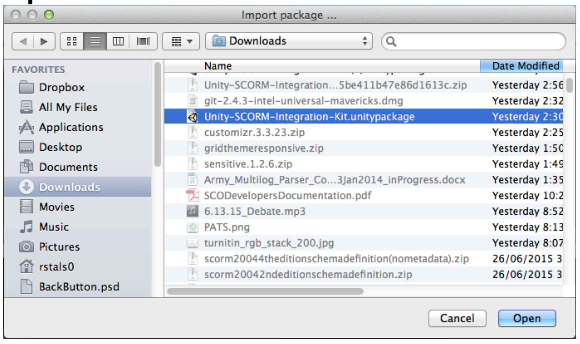
Import the Unity-SCORM Integration Kit

1. Download Unity-SCORM-Integration-Kit.unitypackage.

2. Open your Unity3D project (or create a new project). On the top menu, select **Assets** > **Import Package** > **Custom Package…**

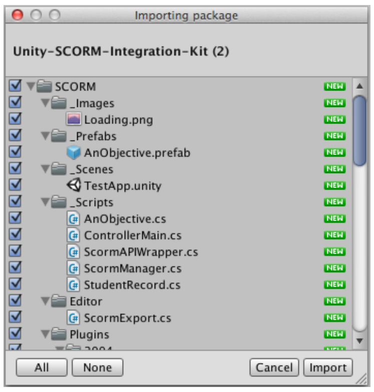


3. Select the Unity-SCORM-Integration-Kit.unitpackage file you downloaded in step 1 and click the **Open** button.

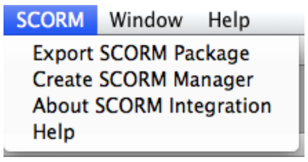


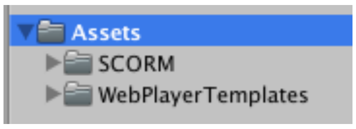
Name	Date Modified
Unity-SCORM-Integration...5be411b47e86d1613c.zip	Yesterday 2:56
git-2.4.3-intel-universal-mavericks.dmg	Yesterday 2:32
Unity-SCORM-Integration-Kit.unitypackage	Yesterday 2:30
customizr.3.3.23.zip	Yesterday 2:25
gridthemeresponsive.zip	Yesterday 1:5
sensitive.1.2.6.zip	Yesterday 1:4
Army_Multilog_Parser_Co...3Jan2014_inProgress.docx	Yesterday 1:35
SCODevelopersDocumentation.pdf	Yesterday 10:2
6.13.15_Debate.mp3	Yesterday 8:52
PATS.png	Yesterday 8:13
turnitin_rgb_stack_200.jpg	Yesterday 8:07
scorm20044theditionschemadefinition(nometadata).zip	26/06/2015 3
scorm20042ndeditionschemadefinition.zip	26/06/2015 3

4. On the Importing package window, click the **Import** button.



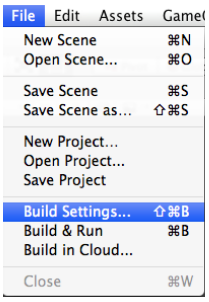
5. The Integration Kit is imported into your project. To refresh the menu, click on any of the menu items. You will then see a new SCORM menu item and two new folders: *SCORM* and *WebPlayer Templates.*



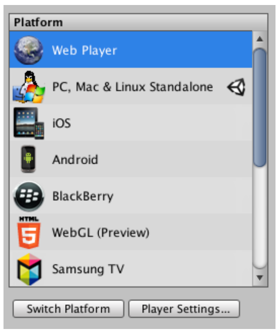


Set up your Unity project

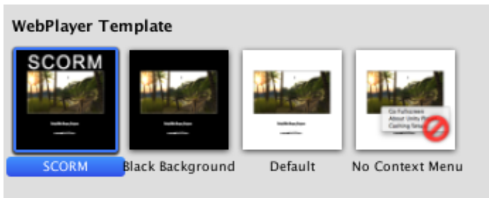
1. Access the *Build Settings*. On the menu select **File** > **Build Settings…**



2. In the *Platform* section, select **Web Player** and then click the **Switch Platform** button.



3. Now click the **Player Settings** button. In the *Inspector window*, change the *WebPlayer Template* to **SCORM.**



You can close the *Build Settings* window.

An Overview of the Assets

1. SCORM
    a. _Images
        i. Loading - The Loading Screen sprite image used in the TestApp scene.
    b. _Prefabs
        i. AnObjective – The Unity3D prefab used whenever a new objective is loaded or created via SCORM. Used in the TestApp Scene.
    c. _Scenes
        i. TestApp – The example SCORM Test Application.
    d. _Scripts
        i. AnObjective.cs – The functionality required to display a SCORM objective on the AnObjective prefab.
        ii. ControllerMain.cs – Drives the Test Application. This script demonstrates how you can use the Integration Kit in your own project.
        iii. ScormAPIWrapper.cs – This acts as the bridge between the SCORM API (served via the scorm.js file that is packaged in the WebPlayer) and Unity3D code.
        iv. ScormManager.cs – The Unity3D side of the SCORM bridge. This contains all the functions needed to load, access and store SCORM data.
        v. StudentRecord.cs – A utility class that simply stores the structure of the data loaded via SCORM.
    e. Editor
        i. ScormExport.cs – The file that drives the SCORM menu item. The most important function is that of exporting and packaging your Unity3D project ready for loading into your LMS.
    f. Plugins
        i. 2004 (folder) – contains the set of files required to created a SCORM 2004 Compliant package. Used by the ScormExport.cs script when creating your package.

   ii. Ionic.Zip.Reduced.dll – A library for creating Zip files. Used by the ScormExport.cs script when creating your package.

   g. Resources

      i. ScormManager – A prefab used to set up a new scene for SCORM integration. Used by the ScormExport.cs script when you select *Create SCORM Manager* from the *SCORM* menu.

2. WebPlayer Templates

   a. SCORM – The SCORM WebPlayet Template to select in the *Build Settings*.

      i. Index.html – the template for the index.html file used when you build this project. Contains the Unity3D Web Player as well as a reference to the scorm.js API file.

      ii. thumbnail – A thumbnail for the Player Settings window.

      iii. scripts

         1. scorm.js – the file that communicates with the LMS via SCORM.

         2. ScormSimulator.js – this can be used to test your own project without having to load it into an LMS first.

Basic SCORM Application Workflow

Setup

To set up a new Unity3D scene ready for SCORM integration, create a new scene and then click the **SCORM** item on the menu, then select **Create SCORM Manager**.

This creates a new GameObject on your scene hierarchy called ScormManager. This has the ScormManager.cs script attached and provides the connection to the SCORM API.

IMPORTANT: Your main controller script that communicates with the ScormManager must be attached to a GameObject that is a child of the ScormManager GameObject. For example, in the Test Application see the ControllerMain GameObject. This is because the ScormManager communicates key events to your code by sending broadcast messages to any GameObjects that are its children.

Workflow

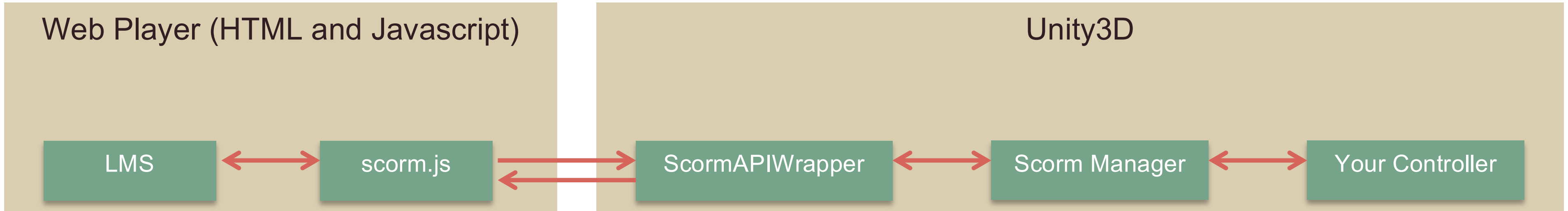
1. On start up, the ScormManager launches a thread that ensures the SCORM API is available in the Web Player and then loads all the available data from the LMS into the StudentRecord.
2. When this is complete, the ScormManager broadcasts the *Scorm_Initialize_Complete* message to your code. Your code should not use the inbuilt Unity3D *Start()* function. Instead, put your initialisation code in a public function called *Scorm_Initialize_Complete()*.
3. It is good practice to set the Completion Status to Incomplete if this is the first attempt (i.e. *StudentRecord.CompletionStatusType.incomplete*). This is particularly important when using your project in Blackboard Learn otherwise Blackboard will set your attempt to Complete by default.
4. Your code then runs. Call the various Get…() functions to read the stored LMS data and call the various Set…() functions to set values to be sent back to the LMS.
5. When complete, set the session time (*ScormManager.SetSessionTime()*), the exit type (*ScormManager.SetExit()*) and then call the Commit function (*ScormManager.Commit ()*).
6. The ScormManager will commit your attempt data to the LMS. When this is finished, it will send the *Scorm_Commit_Complete* message. Do not quit or close your application until your code receives the *Scorm_Commit_Complete* message as you may stop the data being written to the LMS completely.
7. Typically, your *Scorm_Commit_Complete* function will simply call the Terminate function in the ScormManager (*ScormManager.Terminate()*). This in turn sends a message to close the browser window.

There is one additional (optional) function that the ScormManager will call in your code. Any logging information can be manipulated by creating a public function called Log
(i.e. *public void Log(string data)*). In the example Test Application, this functions displays the log data in a text field on the right-hand side of the screen.

Unity3d University

Unity3d.stals.com.au

Architecture Overview



The scorm.js file communicates with the LMS directly.

It performs functions such as:

- Initialise the API connection
- Call getValue() function calls
- Call setValue() function calls
- Commit() data to the LMS
- Terminate() the API connection

Unity3D's communication to and from javascript is asynchronous.

The Scorm API Wrapper forms a bridge between Unity3D and the scorm.js script that appears synchronous to the calling code.

The Scorm Manager provides the interface between your own code and the SCORM API (via the ScormAPIWrapper and the scorm,js script).

It provides functions to load and store data to and from your LMS via SCORM.

This is the main controller that makes use of the functionality provided by the Scorm Manager.

Functions

Full documentation is available at the Unity3D University website: http://unity3d.stals.com.au/Kit-Documentation/index.html.

Key Functions for Your Code

```
public void Scorm_Initialize_Complete()
```

On start-up, the ScormManager ensures the SCORM API is available and then loads all the available data from the LMS. Once this process is complete, it sends a Broadcast Message to your code. You respond to this message by including a public function called Scorm_ Initialize _Complete in your code.

This function should contain the initialisation logic for your project. Do not use the inbuilt Unity3D void Start() function as this may fire before the SCORM API is initialised.

```
public void Log(string data)
```

Most logging that occurs in the Scorm utility classes is passed on to your code via a Broadcast Message. You can respond to logging requests by including this public function in your code.

This function is useful for developing and troubleshooting but should not normally be presented to the end user.

```
UnityEngine.Application.ExternalCall("DebugPrint",data)
```

A related function available to your code is being able to print debug log information to the surrounding HTML window in the Web Player. Simply call the above function, setting data to the string of the data you want logged. This is useful to display error messages to the end user where a crash in your code would prevent the error message being displayed in your own application.

public void Scorm_Commit_Complete()

After you call ScormManager.Commit (), the ScormManager finalises the storing of data to the LMS. Once this process has completed, it sends a Broadcast Message to your code. You respond to this message by including a public function called Scorm_Commit_Complete in your code.

This function should contain any final logic to shut down your application. Typically, the final part of this functions calls ScormManager.Terminate (). This in turn sends a message to the Web Player window to close.

Get Data from the LMS

Learner Data

ScormManager.GetLearnerId ()

cmi.learner_id

Returns the string of the Learner ID set in the LMS.

e.g. string learnerId = ScormManager.GetLearnerId ();

ScormManager.GetLearnerName ()

cmi.learner_name

Returns the string of the Learner Name set in the LMS.

e.g. string learnername = ScormManager. GetLearnerName ();

ScormManager.GetLearnerPreference ()

cmi.learner_preference.X

Returns the StudentRecord.LearnerPreference object.

The StudentRecord.LearnerPreference consists of:

- float audioLevel (cmi.learner_preference.audio_level)
- string language (cmi.learner_preference.language)
- float deliverySpeed (cmi.learner_preference.delivery_speed)
- int audioCaptioning (cmi.learner_preference.audio_captioning)

e.g.StudentRecord.LearnerPreference learnerPreference = ScormManager.GetLearnerPreference ();

float audioLevel = learnerPreference .audioLevel;

ScormManager.GetCommentsFromLearner ()

cmi.comments_from_learner.X

Returns a list of comments - List<StudentRecord.CommentsFromLearner>

The StudentRecord.CommentsFromLearner consists of:

- string comment (cmi.comments_from_learner.n.comment)
- string location (cmi.comments_from_learner.n.location)
- DateTime timestamp (cmi.comments_from_learner.n.timestamp)

e.g.
```
List<StudentRecord.CommentsFromLearner> commentsFromLearnerList =
                                        ScormManager.GetCommentsFromLearner ();

foreach (StudentRecord.CommentsFromLearner comment in commentsFromLearnerList) {
   string timeStamp = comment.timeStamp.ToString();
   string location = comment.location;
   string text = comment.comment;
   …
}
```

SCORM / LMS Data

ScormManager.GetVersion ()

cmi._version

Returns the string of the API version being used.

e.g. string version = ScormManager.GetVersion ();

ScormManager.GetCredit ()

cmi.credit

Returns the StudentRecord.CreditType of this attempt.

e.g. StudentRecord.CreditType credit = ScormManager.GetCredit (); //or

string credit ScormManager.GetCredit ().ToString();

ScormManager.GetEntry()

cmi.entry

Returns the StudentRecord.EntryType of this attempt.

e.g. StudentRecord.EntryType entry = ScormManager. GetEntry (); //or

string entry ScormManager. GetEntry ().ToString();

ScormManager.GetMode()

cmi.mode

Returns the StudentRecord.ModeType of this attempt.

e.g. StudentRecord.ModeType entry = ScormManager. GetMode (); //or

string mode ScormManager. GetMode ().ToString();

ScormManager.GetLocation ()

cmi.location

Returns the string of a previously set Location (bookmark).

e.g. string location = ScormManager.GetLocation ();

ScormManager.GetMaxTimeAllowed ()

cmi.max_time_allowed

Returns the float of the Maximum Time Allowed for this attempt in seconds.

This value is set in the imsmanifest.xml file.

e.g. float maxTimeAllowed = ScormManager.GetMaxTimeAllowed ();

ScormManager.GetTotalTime ()

cmi.total_time

Returns the float of the Total Time spent on this attempt so far in seconds.

e.g. float totalTime = ScormManager. GetTotalTime ();

ScormManager.GetTimeLimitAction ()

cmi.time_limit_action

Returns the StudentRecord.TimeLimitActionType of this SCO.

e.g. StudentRecord.TimeLimitActionType timeLimitAction = ScormManager. GetTimeLimitAction ();
//or

string timeLimitAction ScormManager.GetTimeLimitAction ().ToString();

ScormManager.GetLaunchData ()

cmi.launch_data

Returns the string of the Launch Data for this SCO.

This value is set in the imsmanifest.xml file.

e.g. string launchData = ScormManager. GetLaunchData ();

ScormManager.GetCommentsFromLMS ()

cmi.comments_from_lms.X

Returns a list of comments - List<StudentRecord.CommentsFromLMS>

The StudentRecord.CommentsFromLMS consists of:

- string comment (cmi.comments_from_lms.n.comment)
- string location (cmi.comments_from_lms.n.location)
- DateTime timestamp (cmi.comments_from_lms.n.timestamp)

e.g. List<StudentRecord.CommentsFromLMS> commentsFromLMSList =

```
                                        ScormManager.GetCommentsFromLMS ();

    foreach (StudentRecord.CommentsFromLMS comment in commentsFromLMSList) {
        string timeStamp = comment.timeStamp.ToString();
        string location = comment.location;
        string text = comment.comment;
        …
    }
```

Score Data

ScormManager.GetScaledPassingScore ()

cmi.scaled_passing_score

Returns the float of the Scaled Passing Score for this SCO.

This value is set in the imsmanifest.xml file.

e.g. float scaledPassingScore = ScormManager.GetScaledPassingScore ();

ScormManager.GetScore()

cmi.score

Returns the StudentRecord.LearnerScore for this attempt.

The StudentRecord.LearnerScore consists of:

- float scaled (cmi.score.scaled)
- float raw (cmi.score.raw)
- float max (cmi.score.max)
- float min (cmi.score.min)

e.g. StudentRecord.LearnerScore score = ScormManager.GetScore();
      float rawScore = score.raw;

ScormManager.GetProgressMeasure()

cmi.progress_measure

Returns the float of the measure of progress for this attempt (between 0 and 1).

e.g. float progressMeasure = ScormManager.GetProgressMeasure();

Objective Data

ScormManager.GetObjectives ()

cmi.objectives.n.X

Returns a list of objectives - List<StudentRecord.Objectives> for this attempt.

The StudentRecord.Objectives consists of:

- string id (cmi.objectives.n.id)
- StudentRecord.LearnerScore score (cmi.objectives.n.score.X)
- StudentRecord.SuccessStatusType successStatus (cmi.objectives.n.success_status)
- StudentRecord.CompletionStatusType completionStatus (cmi.objectives.n.completion_status)
- float progressMeasure (cmi.objectives.n.progress_measure)
- string description (cmi.objectives.n.description)

e.g.

```
List<StudentRecord.Objectives> objectivesList = ScormManager.GetObjectives ();
      foreach (StudentRecord.Objectives objective in objectivesList) {
         string id = objective.id;

         …
      }
```

Interaction Data

ScormManager.GetInteractions ()

cmi.interactions.n.X

Returns the list of List<StudentRecord.LearnerInteractionRecord>.

The StudentRecord.LearnerInteractionRecord consists of:

- string id (cmi.interactions.n.id)
- StudentRecord.InteractionType type (cmi.interactions.n.typ)
- DateTime timestamp (cmi.interactions.n.timestamp)
- float weighting (cmi.interactions.n.weighting)
- string response (cmi.interactions.n.response)
- float latency (cmi.interactions.n.latency)
- string description (cmi.interactions.n.description)
- StudentRecord.ResultType result (cmi.interactions.n.result)
- float estimate (cmi.interactions.n.result if it is numeric)
- List<StudentRecord.LearnerInteractionObjective> objectives (cmi.interactions.n.objectives.n.X)
- List<StudentRecord.LearnerInteractionCorrectResponse> correctResponses (cmi.interactions.n.correct_responses.n.X)

e.g.

```
List<StudentRecord.LearnerInteractionRecord> interactionsList = ScormManager.GetInteractions ();
    foreach (StudentRecord.LearnerInteractionRecord record in interactionsList) {
        string id = record.id;
        string type = record.type.ToString();
        …
}
```

Exit Data

ScormManager.GetCompletionThreshold ()

cmi.completion_threshold

The float of the Completion Threshold for this SCO.

This is set in the imsmanifest.xml file.

e.g. float completionThreshold = ScormManager.GetCompletionThreshold ();

ScormManager.GetSuccessStatus()

cmi.success_status

The StudentRecord.SuccessStatusType for this attempt.

e.g. StudentRecord.SuccessStatusType successStatus = ScormManager.GetSuccessStatus (); //or

```
if(successStatus == StudentRecord.SuccessStatusType.passed)
        …
```

ScormManager.GetCompletionStatus()

cmi.completion_status

The StudentRecord.CompletionStatusType for this attempt.

e.g.

StudentRecord.CompletionStatusType completionStatus = ScormManager.GetCompletionStatus (); //or

```
if(completionStatus == StudentRecord.CompletionStatusType.completed)
        …
```

Send Data to the LMS

Use these functions to send data back to the LMS. The data is not 'saved' to the LMS until a successful call to ScormManager.Commit() is completed.

Learner Data

ScormManager.SetLearnerPreferenceAudioCaptioning (float value)

cmi.learner_preference.audio_captioning

ScormManager. SetLearnerPreferenceAudioLevel (float value)

cmi.learner_preference.audio_level

ScormManager. SetLearnerPreferenceDeliverySpeed (float value)

cmi.learner_preference.delivery_speed

ScormManager. SetLearnerPreferenceLanguage (string value)

cmi.learner_preference.language

Set the value of the various Learner Preference items.

ScormManager.AddCommentFromLearner(StudentRecord.CommentsFromLearner comment)

cmi.comments_from_learner.X

Add a new comment from the Learner.

e.g.

```
StudentRecord.CommentsFromLearner comment = new StudentRecord.CommentsFromLearner();
comment.comment = commentText;
comment.location = commentLocation;
ScormManager.AddCommentFromLearner(comment);
```

SCORM / LMS Data

ScormManager.SetSessionTime (float value)

cmi.session_time

Set the session time for this attempt in seconds.

ScormManager.SetLocation(string value)

cmi.location

Set the location (bookmark) of the student’s attempt when the SCO is suspended.

You should exit the application with a status of suspend immediately after setting a location.

e.g.

```
ScormManager.SetSessionTime (currentTime);

ScormManager.SetLocation (“Chapter 1”);

ScormManager.SetExit (StudentRecord.ExitType.suspend);

ScormManager.Commit ();
```

Score Data

ScormManager.SetProgressMeasure(float value)

cmi.progress_measure

ScormManager.SetScoreScaled(float value)

cmi.score.scaled

ScormManager.SetScoreMin(float value)

cmi.score.min

ScormManager.SetScoreMax(float value)

cmi.score.max

ScormManager.SetScoreRaw(float value)

cmi.score.raw

Set the various items related to the student’s score and progress.

Objectives Data

ScormManager.AddObjective(StudentRecord.Objectives objective)

cmi.objectives.n.X

Add a new objective. If you associate an objective with an Interaction, you must first add an objective.

e.g.

```
StudentRecord.Objectives newObjective = new StudentRecord.Objectives();
newObjective.id = “A unique ID”;
newObjective.description = “Description of the objective”;

StudentRecord.LearnerScore score = new StudentRecord.LearnerScore();
score.min = 0f;
score.max = 100f;
score.raw = 0f;
score.scaled = 0f;

newObjective.score = score;
newObjective.successStatus = StudentRecord.SuccessStatusType.unknown;
newObjective.completionStatus = StudentRecord.CompletionStatusType.not_attempted;
newObjective.progressMeasure = 0f;

ScormManager.AddObjective(newObjective);
```

Interactions Data

ScormManager.AddInteraction(StudentRecord.LearnerInteractionRecord interaction)

cmi.interactions.n.X

Add a new interaction from the student.

e.g.

```
StudentRecord.LearnerInteractionRecord newInteraction = new StudentRecord.LearnerInteractionRec
ord ();
newInteraction.id = ScormManager.GetNextInteractionId();
newInteraction.timeStamp = DateTime.Now;
newInteraction.type = StudentRecord.InteractionType.other;
newInteraction.weighting = 1f;
newInteraction.response = “What the student wrote / chose / did”;
newInteraction.latency = 18.4f;
newInteraction.description = “A description of the interaction / question”;
StudentRecord.ResultType result = StudentRecord.ResultType.incorrect;
newInteraction.result = result;
```

```
ScormManager.AddInteraction (newInteraction);
```

Exit Data

ScormManager.SetSuccessStatus(StudentRecord.SuccessStatusType value)

cmi.success_status

ScormManager.SetCompletionStatus(StudentRecord.CompletionStatusType value)

cmi.completion_status

Set the Success and Completion status for this attempt.

e.g.

```
ScormManager.SetCompletionStatus (StudentRecord.CompletionStatusType.completed);
ScormManager.SetSuccessStatus (StudentRecord.SuccessStatusType.passed);
```

ScormManager.SetExit(StudentRecord.ExitType value)

cmi.exit

Set the exit type of this attempt.

e.g.

```
ScormManager.SetSessionTime (currentTime);
ScormManager.SetExit (StudentRecord.ExitType.normal);
ScormManager.Commit ();
```