

agrega⁺

Objeto digital (SCORM)

Guía práctica de creación de SCO



red.es

plan
avanza



Comunidades
Autónomas

GUÍA PRÁCTICA DE CREACIÓN DE SCO'S

1. FUNCIONES BÁSICAS

Un script mínimo debe proporcionar funciones para inicializar y terminar la sesión de comunicación automáticamente si existe una instancia de la API de SCORM 2004. Además debe poner el estado de terminación del SCO a "completado" cuando el SCO termina. Así en este script se han implementado 4 funciones:

- ScanForAPI(win)
Esta función busca la existencia de una instancia de la API de SCORM 2004 en el sistema.
- GetAPI(win)
Esta función recupera una instancia de la API de SCORM 2004, en caso de existir en el sistema.
- ScormInitialize()
Esta función inicializa la comunicación con una instancia de la API de SCORM 2004.
- ScormTerminate()
Esta función finaliza la comunicación con una instancia de la API de SCORM 2004.

```
// For SCORM 2004 only
var gAPI = null;
var gnScormSessionState = 0; // 0-not initialized; 1-initialized; 2-terminated
function ScanForAPI(win)
{
    var nFindAPITries = 500;
    while ((win.API_1484_11 == null) && (win.parent != null) && (win.parent != win))
    {
        nFindAPITries--;
        if (nFindAPITries < 0) return null;
        win = win.parent;
    }
    return win.API_1484_11;
}
function GetAPI(win)
{
    if ((win.parent != null) && (win.parent != win))
    {
        gAPI = ScanForAPI(win.parent);
    }
    if ((gAPI == null) && (win.opener != null))
    {
        gAPI = ScanForAPI(win.opener);
    }
}
function ScormInitialize()
{
    if (gnScormSessionState == 0)
    {
        GetAPI(window);
        if ((gAPI != null) && (gAPI.Initialize("") == "true"))
        {
            gnScormSessionState = 1;
        }
    }
}
function ScormTerminate()
{
    if (gnScormSessionState == 1)
    {
        gAPI.SetValue("cm1.completion_status", "completed")
        if (gAPI.Terminate("") == "true") gnScormSessionState = 2;
    }
}
```

Disponiendo de este script se puede crear un SCO que use las funciones del mismo como el siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Claude's simplest SCO</title>
<script type="text/javascript" src="minisco.js">
</script>
</head>
<body onload=ScormInitialize() onunload=ScormTerminate()>
<em>... as simple as possible, but not simpler.</em><br />Albert Einstein
</body>
</html>
```

En este SCO destacan las siguientes líneas de código:

a) <script type="text/javascript" src="minisco.js">

En esta línea de código se hace referencia al script que contiene las funciones que va a usar el SCO.

b) <body onload=ScormInitialize() onunload=ScormTerminate()>

En esta línea mediante la asignación **onload=ScormInitialize()**, se asegura que el SCO empezará automáticamente una sesión de comunicación con la API de SCORM tan pronto como se cargue llamando a la función ScormInitialize(). Y con la asignación **onunload=ScormTerminate()**, se asegura que si el SCO finaliza la ejecución de forma inesperada, entonces se realizará un cierre ordenado que permita enviar los datos de la sesión (tracking) no grabados a través de la API de SCORM y terminar la sesión de comunicación.

2. INTERCAMBIO DE DATOS

El script anterior se puede completar con nuevas funciones para poder llevar a cabo un intercambio de datos entre el SCO y el entorno de ejecución.

```
function ScormGetLastError()
{
    var sErr = "-1";
    if (gAPI) sErr = gAPI.GetLastError();
    return sErr;
}
function ScormGetErrorString(sErr)
{
    var strErr = "SCORM API not available";
    if (gAPI)
    {
        // Note: Get Error functions may work even if the session is not open
        // (to help diagnose session management errors), but we're still careful,
        // and so we check whether each function is available before calling it.
        if ((isNaN(parseInt(sErr))) && (gAPI.GetLastError()) sErr = gAPI.GetLastError());
        if (gAPI.GetErrorText) strErr = gAPI.GetErrorText(sErr.toString());
    }
    return strErr;
}
function ScormGetValue(what)
{
    var strR = "";
    if (gnScormSessionState == 1)
    {
        strR = gAPI.GetValue(what);
        if ((strR == "") && (ScormGetLastError() != 0)) alert(ScormGetErrorString());
    }
    return strR;
}
function ScormSetValue(what, value)
{
    if (gnScormSessionState == 1)
    {
        return gAPI.SetValue(what, value);
    }
    else
    {
        return "false";
    }
}
```

Así en este script se han implementado 4 funciones:

- ScormGetLastError()
Esta función recupera un código que representa el estado de error de la sesión de comunicación después de la última llamada a la API.
- ScormGetErrorString(Estado de Error)

agrega

Esta función recupera el mensaje de error asociado a un estado error determinado especificado en sErr.

- ScormGetValue(Elemento de datos)
Esta función permite recuperar datos almacenados en el entorno de ejecución, hace uso de las funciones ScormGetLastError para detectar si se ha producido algún error, y de ScormErrorString para mostrar mensajes de error en caso de haberse producido alguno. Para ello toma como parámetros el elemento de datos del modelo CMI que se quiere consultar.
- ScormSetValue(Elemento de datos,Valor)
Esta función permite enviar datos para su almacenamiento en el entorno de ejecución. Para ello toma como parámetros el elemento de datos del modelo CMI que se quiere actualizar, y el valor con el que se quiere actualizar(se trata de valores predefinidos para cada elemento de datos).

Observar que se trata de funciones genéricas, desde las que se llama a la API. Esta forma de implementación permite aislar al SCO de cambios que puedan producirse en la API de SCORM.

Considerando las nuevas funciones, y una modificación de la función ScormTerminate():

```
function ScormTerminate()
{
  if (gnScormSessionState == 1)
  {
    if (gAPI.Terminate("") == "true")
    {
      gnScormSessionState = 2;
      return "true";
    }
  }
  return "false";
}
```

se puede crear el siguiente script:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Claude's simplest SCO</title>
<script type="text/javascript" src="simplesco.js"></script>
<script type="text/javascript">
function init()
{
  ScormInitialize();
  if (ScormGetValue("cm1.completion_status") != "completed")
  {
    ScormSetValue("cm1.completion_status","incomplete");
  }
}
function terminate()
{
  ScormSetValue("cm1.completion_status","completed");
  ScormTerminate();
}
</script>
</head>
<body onload="init()" onunload="terminate()">
<em>Perfection is achieved not when there is nothing left to add, but when
there is nothing left to remove.</em><br />-- Antoine de Saint-Exup&eacute;ry
</body>
</html>
```

En este SCO cabe destacar los siguientes trozos de código:

- a) <script type="text/javascript" src="simplesco.js">
En esta línea de código se hace referencia al script que contiene las funciones que va a usar el SCO.
- b)

agrega

```
<script type="text/javascript">
function init()
{
  ScormInitialize();
  if (ScormGetValue("cm1.completion_status") != "completed")
  {
    ScormSetValue("cm1.completion_status","incomplete");
  }
}
function terminate()
{
  ScormSetValue("cm1.completion_status","completed");
  ScormTerminate();
}
</script>
```

En esta línea de código se definen dos funciones que se van a usar directamente en el SCO:

- init()

Permite inicializar la comunicación del SCO inmediatamente a continuación de que el SCO es lanzado. A continuación pone el estado del SCO a "incompleto", en caso de que el entorno de ejecución no hubiera reportado que el estado actual del SCO es "completado".

- terminate()

Permite finalizar la comunicación estableciendo el estado del SCO a "completado", antes de terminar la sesión de comunicación.

Observar que se ha eliminado la línea de código de ScormTerminate() que hacia la actualización del estado del SCO a "completado" cuando éste iba a finalizar, y se ha insertado esta acción, dentro de la nueva función terminate() definida directamente en el SCO.

c) <body onload="init()" onunload="terminate()">

En esta línea mediante la asignación **onload=init()**, se asegura que el SCO empezará automáticamente una sesión de comunicación con la API de SCORM tan pronto como se cargue llamando a la función init(). Y con la asignación **onunload=terminate()**, se asegura que si el SCO finaliza la ejecución de forma inesperada, entonces se realizará un cierre ordenado que permita enviar los datos de la sesión (tracking) no grabados a través de la API de SCORM y terminar la sesión de comunicación.

Otro SCO más complejo que usa estas funciones, y que define otras internamente sería:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Claude's simple tracking SCO</title>
<script type="text/javascript" src="simplesco.js"></script>
<script type="text/javascript">
function init()
{
    ScormInitialize();
    if (ScormGetValue("cm1.completion_status") != "completed")
    {
        ScormSetValue("cm1.completion_status","incomplete");
    }
}
function terminate()
{
    ScormSetValue("cm1.completion_status","completed");
    ScormSetValue("cm1.progress_measure","1.0");
    ScormSetValue("cm1.success_status","passed");
    ScormSetValue("cm1.score.scaled","1.0");
    ScormTerminate();
}
</script>
</head>
<body onload="init()" onunload="terminate()">
<em>Perfection is achieved not when there is nothing left to add, but when
there is nothing left to remove.</em><br />-- Antoine de Saint-Exup&eacute;ry
</body>
</html>

```

Un SCO puede reportar a un LMS dos tipos de informaciones:

- Información de progreso, que puede ser a su vez de dos tipos:
 - o Estado de completitud
Toma el valor de completado o no completado.
 - o Medida del progreso
Es un ratio entre lo que está realizado y lo que puede ser realizado, y se representa como un valor en coma flotante entre 0 (nada realizado) y 1 (completamente realizado).
- Información del éxito, que puede ser a su vez de dos tipos:
 - o Estado del éxito
Toma el valor de aprobado o fallado.
 - o Medida del éxito
Es un valor escalado en el rango entre -1.0 y 1.0, donde 0 representa no éxito, 1.0 representa éxito total, y los valores negativos representan penalizaciones.

Además de las puntuaciones escaladas, también podría reportarse información sobre el rango de puntuación, como por ejemplo la máxima y mínima puntuación que define el rango para una fila de puntuaciones. Sin embargo, en la práctica, sólo se usan las puntuaciones escaladas, mostrándose estas puntuaciones como valores porcentuales. Por otro lado señalar, que SCORM no establece un conjunto de datos de seguimiento predeterminado. En este SCO, los valores se insertan directamente, sin embargo en un caso más real, estos valores se obtendrían en función de la interacción del usuario que usa el SCO. Otra observación importante es que el SCO no envía valores mínimos, máximos o filas de puntuaciones dado que los LMSs no los tienen en cuenta.

Por otro lado un SCO puede ser usado en diferentes contextos, mediante el uso de diferentes valores umbrales que determinarían la puntuación de aprobado o suspenso. Para ello el SCO podría consultar al entorno de ejecución por la existencia de esos valores, y en caso de no existir, usar los suyos propios. Por ejemplo el SCO anterior podría ser usado en otros contextos si se modifica la función ScormSetValue del script, de forma que tenga en cuenta el valor umbral de éxito para actualizar el estado de éxito cuando una puntuación es reportada. Si no existe ningún valor umbral en el entorno de ejecución se usaría el del propio SCO.

```
// script fragment
var gnPassingScore = 1.0; // Default value for this SCO
var gbPassingScoreAlreadyQueriedFromRTE = false; // True if RTE queried for value
var gbPassingScoreIsFromRTE = false; // True only if RTE did provide value

function ScormSetValue(what, value)
{
    var err = "false"
    if (gnScormSessionState == 1)
    {
        err = gAPI.SetValue(what, value);
        if ((err == "true") && (what == "cm1.score.scaled"))
        {
            ScormSetSuccessStatusForScore(parseFloat(value));
        }
    }
    return err;
}

function ScormSetSuccessStatusForScore(nScore)
{
    if (!gbPassingScoreAlreadyQueriedFromRTE)
    {
        var nThreshold = parseFloat(ScormGetValue("cm1.scaled_passing_score"));
        gbPassingScoreAlreadyQueriedFromRTE = true;
        if (IsValidScaledScore(nThreshold))
        {
            gnPassingScore = nThreshold;
            gbPassingScoreIsFromRTE = true;
        }
    }
    if (IsValidScaledScore(nScore) && (IsValidScaledScore(gnPassingScore)))
    {
        (nScore >= gnPassingScore? ScormSetValue("cm1.success_status", "passed"):
        ScormSetValue("cm1.success_status", "failed"));
    }
}

function IsValidScaledScore(nScore)
{
    return ((!isNaN(nThreshold)) && (nThreshold >= -1.0) && (nThreshold <= 1.0))
}

```

3. MANTENIMIENTO DEL ESTADO

Cuando un SCO está formado por más de una página HTML, se presenta el problema de mantener el estado de la sesión de la comunicación y el valor de determinadas variables, entre las diferentes páginas HTML, dado que por cada lanzamiento de SCO, existe una única sesión de comunicación, no pudiendo tener inicializada y terminada las sesiones de comunicación de cada una de ellas. Hay dos formas de superar este problema:

- Implementar el SCO como un frameset invisible dentro del cual se cambian las páginas sin perder los datos de estado almacenado en él. Básicamente consiste en cambiar la fuente del frame "stage" en el frameset. Además el frameset puede actuar como un entorno de ejecución o secuenciación propio para las páginas, gestionando la navegación mediante un script, que disponga de funciones que hacen uso de llamadas desde las páginas a la instancia de la API provista por el entorno de ejecución de SCORM. La única consideración a tener en cuenta es la accesibilidad del frameset, para lo que es necesario que el "stage" frame tenga un nombre principal y que cada página tenga un título principal. Por otro lado el título de cada página será invisible en un browser normal, puesto que la página es mostrado en un frame sin adornos, pero es importante proporcionar esto así para los browsers accesibles.
- Usar cookies.

El siguiente SCO usa el mismo script genérico que el ejemplo anterior, sin embargo no se pone el estado de completitud a "completado" hasta que se han visitado todas las páginas. También implementa algunas funciones que proporcionan navegación entre las páginas.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Claude's simplest multiple page SCO</title>
<script type="text/javascript" src="simplesco.js"></script>
<script type="text/javascript">
function init()
{
    ScormInitialize();
    if (ScormGetValue("cm1.completion_status") != "completed")
    {
        ScormSetValue("cm1.completion_status","incomplete");
    }
}
function terminate()
{
    MarkIfCompleted();
    ScormTerminate();
}
var gnPage = 1;
var gnMaxPages = 3; // Change this if you add more pages
var gnPagesNeeded = gnMaxPages; // Change this if fewer pages are needed for
completion
var gaPagesCompleted = new Array(true,false); // Used to keep track of pages viewed
for (i=1; i < gnMaxPages; i++) gaPagesCompleted[i] = false;
function GoToPage(n)
{
    if (gnPage != n)
    {
        gnPage = n;
        DisplayFrame.location.href = "page" + gnPage + ".html";
        gaPagesCompleted[gnPage-1] = true;
        MarkIfCompleted();
    }
}
function GoPreviousPage()
{
    if (gnPage > 1) GoToPage(gnPage - 1);
}
function GoNextPage()
{
    if (gnPage < gnMaxPages) GoToPage(gnPage + 1);
}
function MarkIfCompleted()
{
    var nCompleted = 0;
    var bCompleted = false;
    for (i=0; i < gaPagesCompleted.length; i++)
    {
        if (gaPagesCompleted[i]) nCompleted++;
    }
    bCompleted = (nCompleted >= gnPagesNeeded)
    if (bCompleted) ScormSetValue("cm1.completion_status","completed");
    return bCompleted;
}
</script>
</head>
<frameset rows="100%,*"
    onload="init()" onunload="terminate()">
    <frame id="DisplayFrame" name="DisplayFrame" src="page1.html" />
    <frame id="DummyFrame" name="DummyFrame" src="about:blank" />
</frameset>
</html>

```

Cada página del SCO contiene los mismos enlaces de navegación básicos tal como se ve en el ejemplo de más abajo. Sin embargo, la página 1 no contendrá el enlace de "Previo" y la última página el enlace de "Next". El chequeo de la función en la página padre no es estrictamente necesario, pero previene de desconcertantes mensajes de error, si la página es usada fuera del contexto del frameset.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Page two</title>
<script type="text/javascript">
function previous()
{
    if ((window.parent)&&(window.parent.GoPreviousPage)) window.parent.GoPreviousPage()
}
function next()
{
    if ((window.parent)&&(window.parent.GoNextPage)) window.parent.GoNextPage()
}
</script>
</head>
<body>
<p>This is page 2</p>
<p><a href="javascript:previous()">Previous</a></p>
<p><a href="javascript:next()">Next</a></p>
</body>
</html>
```

4. PERSISTENCIA DE LA INFORMACIÓN

No existe una forma estándar para que el entorno de ejecución almacene los datos persistentes, a menos que el SCO se lo pida llamando a la función Commit de la instancia del API. Un buen momento para hacer esto es cuando se va a grabar un cambio en la página del frameset. Para implementar esta solución se llevan a cabo:

1) Añadir una función genérica Commit al script, de forma que esta función permita añadir un manejador de error y depurador de código.

```
...
function ScormCommit()
{
    if ((gnScormSessionState -- 0) && (API != null))
    {
        return gAPI.Commit("");
    }
    return "false";
}
...
```

2) Añadir la llamada Commit si se almacena la compleción mientras el SCO está ejecutándose.

En el siguiente ejemplo se muestra como se puede añadir el uso de la función commit, en la función GoToPage del script del frameset, para ser llamada si se produce la compleción del SCO en la función.

```
...
function GoToPage(n)
{
    if (gnPage != n)
    {
        gnPage = n;
        DisplayFrame.location.href = "page" + gnPage + ".html";
        gaPagesCompleted[gnPage-1] = true;
        if (MarkIfCompleted()) ScormCommit();
    }
}
...
```

Observar que la llamada Terminate implícitamente llama a Commit, de forma que en un SCO no es necesaria la llamada a Commit si una o más llamadas a SetValue van seguidas inmediatamente por una llamada a Terminate.

5. SUSPENDER Y REANUDAR

Sería interesante que el usuario pudiera volver a la misma página si al intentar finalizar la actividad que usa el SCO es interrumpido y entonces se reanuda más tarde. Esto requiere que el SCO notifique al entorno de ejecución que quiere que la sesión sea suspendida. El SCO puede entonces almacenar algunos datos específicos en los elementos de datos `cmi.location` y `cmi.suspend_data`, y si la sesión es más tarde reanudada, podrá usar los datos almacenados para reanudar su propio estado. Para implementar esta solución no hay que modificar el script genérico, pero si es necesario modificar el script inserto en el SCO para grabar la información de estado e invocar la suspensión antes de la terminación. Además de esta forma se puede detectar si se está reanudando una sesión inicial cuando es lanzado. Solo el SCO comprende los valores de los datos en `cmi.location` y `cmi.suspend_data`. En este sentido el entorno de ejecución y el LMS no hacen ni deberían intentar interpretar los valores de esos elementos de datos. Por otra parte tendría sentido validar los valores que el SCO consigue del entorno de ejecución antes de usarlos. Los datos suspendidos y localizados pueden que necesiten ser convertidos a una cadena que sea enviada a través de la API, y sea devuelta en otro modelo datos cuando se reciba de vuelta a través de la API. En este ejemplo, la localización es un número, y el dato suspendido es usado para restaurar un array. Dado que este SCO siempre quiere ser suspendido si es posible, y nos gustaría reanudarlo donde se dejó en caso de un fallo, entonces se actualizan los datos suspendidos cuando el usuario va de una página a otra, en vez de esperar al último momento cuando el SCO detecta que va a ser descargado.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Claude's simplest multiple page SCO</title>
<script type="text/javascript" src="simpleSCO.js"></script>
<script type="text/javascript">
function init()
{
    ScormInitialize();
    if (ScormGetValue("cmi.completion_status") != "completed")
    {
        ScormSetValue("cmi.completion_status", "incomplete");
    }
    if (ScormGetValue("cmi.entry") == "resume") RestoreFromSuspend();
}
function terminate()
{
    MarkIfCompleted();
    ScormTerminate();
}
var gnPage = 1;
var gnMaxPages = 3; // Change this if you add more pages
var gnPagesNeeded = gnMaxPages; // Change if fewer pages are needed for completion
var gaPagesCompleted = new Array(true,false); // Used to keep track of pages viewed
for (i=1; i < gnMaxPages; i++) gaPagesCompleted[i] = false;
function SaveSuspendData()
{
    ScormSetValue("cmi.exit","suspend");
    ScormSetValue("cmi.location", gnPage.toString()); //SCORM requires string type
    ScormSetValue("cmi.suspend_data", gaPagesCompleted.join(","))
}
```

```

function RestoreFromSuspend()
{
    var a = ScormGetValue("cmi.suspend_data").split(",");
    if (a.length == gaPagesCompleted.length) gaPagesCompleted = a;
    var n = parseInt(ScormGetValue("cmi.location"));
    if (!isNaN(n)) GoToPage(n)
}
function GoToPage(n)
{
    if (gnPage != n)
    {
        gnPage = n;
        DisplayFrame.location.href = "page" + gnPage + ".html";
        gaPagesCompleted[gnPage-1] = true;
        MarkIfCompleted();
    }
}
function GoPreviousPage()
{
    if (gnPage > 1) GoToPage(gnPage - 1);
}
function GoNextPage()
{
    if (gnPage < gnMaxPages) GoToPage(gnPage + 1);
}
function MarkIfCompleted()
{
    var nCompleted = 0;
    var bCompleted = false;
    for (i=0; i < gaPagesCompleted.length; i++)
    {
        if (gaPagesCompleted[i]) nCompleted++;
    }
    bCompleted = (nCompleted >= gnPagesNeeded)
    if (bCompleted) ScormSetValue("cmi.completion_status", "completed");
    return bCompleted;
}
</script>
</head>
<frameset rows="*,100%"
onload="init()" onunload="terminate()">
    <frame id="DummyFrame" name="DummyFrame" src="about:blank" />
    <frame name="DisplayFrame" name="DisplayFrame" src="page1.html" />
</frameset>

```

6. INFORMACIÓN QUE SE REPORTA

SCORM 2004 permite a un SCO crear hasta 250 registros de interacción. Cada registro contiene un identificador para esa interacción. Durante una sesión de comunicación, el entorno de ejecución almacena los registros en un array indexado. El array de interacciones solo persiste durante la sesión de comunicación y el entorno de ejecución puede usar distintas aproximaciones para almacenar estos registros entre sesiones. La colección de registros de interacción no se encuentra almacenado en ningún orden determinado. Debido a esto último, en una comunicación posterior, los registros de interacción podrían aparecer en un orden diferente. En este sentido si se van a usar de una sesión previa, habría que encontrar que índice del array se le asignó en la nueva sesión, para lo cual se buscará los registros por el identificador de interacción.

Para gestionar esto se añaden nuevas funciones al script de SCORM:

```
// fragment
function ScormInteractionAddRecord (strID, strType)
{
  var n = ScormInteractionGetIndex(strID);
  if (n > -1) // An interaction record exists with this identifier
  {
    if (ScormGetValue("cmi.interactions." + n + ".type") != strType) return -1;
    return n;
  }
  n = ScormInteractionGetCount();
  var strPrefix = "cmi.interactions." + n + ".";
  if (ScormSetValue(strPrefix + "id", strID) != "true") return -1;
  if (ScormSetValue(strPrefix + "type", strType) != "true") return -1;
  return n
}
function ScormInteractionGetCount ()
{
  var r = parseInt(ScormGetValue("cmi.interactions.count"));
  if (isNaN(r)) r = 0;
  return r;
}
function ScormInteractionGetData(strID, strElem)
{
  var n = ScormInteractionGetIndex(strID);
  if (n < 0)
  {
    return ""; // No interaction record exists with this identifier
  }
  return ScormGetValue("cmi.interactions." + n + "." + strElem);
}
function ScormInteractionGetIndex(strID)
{
  var n = ScormInteractionCount();
  for (i = 0; i < n; i++)
  {
    if (ScormGetValue("cmi.interactions." + i + ".id") == strID)
    {
      return i;
    }
  }
  return -1;
}
function ScormInteractionSetData(strID, strElem, strVal)
{
  var n = ScormInteractionGetIndex(strID);
  if (n < 0)
  {
    return "false"; // No interaction record exists with this identifier
  }
  return ScormSetValue("cmi.interactions." + n + "." + strElem, strVal);
}
```

- ScormInteractionAddRecord toma como parámetro el identificador de una interacción y un tipo de interacción válida. Retorna un entero que corresponde al índice del array de registros o -1 si ocurre algún error. Si un registro de interacción con el mismo identificador y el mismo tipo ya existe, la función no hace nada y retorna el índice del registro existente. Si un registro de interacción con el mismo identificador pero con diferente tipo ya existe, la función falla. Si el número de registros de interacción permitidos se excediera por la adición de un nuevo registro, la función fallaría.
- ScormInteractionGetCount no toma ningún parámetro y retorna un valor entero que representa el número de registros de interacción existentes.
- ScormInteractionGetData toma como parámetro el identificador de la interacción y el elemento del modelo de datos dentro del registro de la interacción, y retorna un valor. Si el valor es una cadena vacía, podría indicar un posible error. La cadena que identifica el elemento de datos es la que aparece a la derecha de "interactions.n.." en la documentación de SCORM para el modelo de datos.
- ScormInteractionGetIndex toma como parámetro el identificador de la interacción. Retorna un entero, que es el índice del registro de la interacción o -1 si el registro no existe. Se puede usar esta función para chequear si existe un registro para esa interacción.

agrega ⁺

- ScormInteractionSetData toma como parámetro el identificador de una interacción, el elemento del modelo de datos dentro del registro de interacción y el valor a poner. Retorna cierto o falso. Si retorna falso, se puede analizar el estado de error. Dado que los datos no pueden ser almacenados en un registro de interacción hasta que el identificador es almacenado en el registro, y algunos datos no pueden ser almacenados apropiadamente a menos que el tipo de interacción sea conocida, la función fallará si el registro de interacción no ha sido creado previamente por una llamada ScormInteractionAddRecord. La cadena que identifica el elemento de datos es la que aparece a la derecha de "interactions.n.." en la documentación de SCORM para el modelo de datos.

Usando esta API se puede crear el siguiente SCO:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>A simple SCO with interaction tracking</title>
<script type="text/javascript" src="ostyn2004sco.js"> </script>
<script type="text/javascript">
function evalQChoice(thisForm, strCorr)
{
  var n = 0;
  var strID = thisForm.id;
  var aResp = new Array();
  for (i = 0; i < thisForm.elements.length; i++)
  {
    if (thisForm.elements[i].checked) aResp[aResp.length] = thisForm.elements[i].id;
  }
  var strResp = aResp.toString();
  (strResp != strCorr ? strResult = "incorrect" : strResult = "correct");
  (strResp != strCorr ? alert("Oops!") : alert("Well done!"));
  ScormInteractionAddRecord(strID, "choice");
  ScormInteractionSetData(strID, "description", "Some sample multiple choice");
  ScormInteractionSetData(strID, "correct_responses.0.pattern", strCorr);
  ScormInteractionSetData(strID, "learner_response", strResp);
  ScormInteractionSetData(strID, "result", strResult);
  ScormCommit();
  return false; // this is a local submit - we don't want to unload the page
}
</script>
</head>
<body><!-- onload and onunload are managed by ostyn2004sco.js -->
<form id="urn_ostyn_q_sa20041101_foo" onsubmit="return evalQChoice(this, 'A1,A3')">
Which of these functions <b>must</b> be used by a SCO to be SCORM conformant?<br />
<input type="checkbox" id="A1" size="50" />Initialize<br />
<input type="checkbox" id="A2" size="50" />Commit<br />
<input type="checkbox" id="A3" size="50" />Terminate<br />
<input type="submit" value="Done" /></form>
</body>
</html>
```

Este SCO hace un seguimiento simple de la interacción, y almacena una puntuación (observar que no existe ningún requisito de que la puntuación este ligado a los resultados de la interacción). Además el SCO incluye un script que gestiona de manera automática los eventos de unload y onunload.

7. DATOS OBJETIVOS

Todo SCO tiene asociado un objetivo implícito, que típicamente es alguna forma de resultados de aprendizaje. La estado con respecto a este objetivo implícito es registrada como datos de estado por el SCO en si mismo: cantidad que se ha completado, y el modo en que el estudiante está teniendo éxito. Sin embargo, un SCO puede gestionar información de estado de otro tipo de objetivos, denominados explícitos, los cuales están identificados por un identificador único. El estado de cada uno de estos objetivos puede ser almacenado en registros de objetivos que incluyen este identificador. El identificador para un objetivo

explícito debería ser globalmente único, dado la secuenciación y navegación hace uso de los mismos para crear mapas de objetivos para actividades. Un mapa de objetivos permite al autor especificar la forma de cómo los datos de estado acerca del objetivo referenciado por el SCO es compartido con otras actividades. Esto permite a un SCO influir sobre los estados de los objetivos que son referenciados por las reglas de secuencia, y también permite encontrar información acerca de los objetivos influidos por otras actividades.

El modelo de datos para interacciones incluye referencias opcionales a los objetivos. Sin embargo estas referencias son puramente informativas. Una referencia a un objetivo en los datos para una interacción, no implica que de algún modo se influya sobre los datos de estado para el objetivo. Si se quiere poner datos de objetivos como resultados de las interacciones en un SCO, se deben incluir en la codificación o scripting del SCO.

SCORM 2004 permite a un SCO acceder o crear hasta 100 registros de estado de objetivos. Cada registro de objetivo debe contener un identificador para ese objetivo. El identificador es el único camino para identificar un registro de objetivo. Durante una sesión de comunicación, el entorno de ejecución almacenará los registros en un array indexado. Cuando se inicia un intento en un SCO, algunos registros de objetivo podrían ya existir en la colección de registros de objetivos para el SCO. Esto ocurrirá cuando los identificadores de objetivos correspondientes son especificados en un mapa de objetivos en las propiedades de secuenciación para la actividad que es lanzada por el SCO. Por otro lado señalar, que la colección de registros de objetivos no está ordenada. Además el array de objetivos solo persiste durante la sesión de comunicación, y el entorno de ejecución podría usar diferentes aproximaciones para almacenar estos registros entre sesiones. Así en sesiones de comunicación subsecuentes, los registros de objetivos podrían aparecer en orden diferente. En este sentido si se van a usar de una sesión previa, habría que encontrar que índice del array se le asignó en la nueva sesión, para lo cual se buscará los registros por el identificador de interacción.

Para gestionar esto se añaden nuevas funciones al script de SCORM:

```
function ScormObjectiveAddRecord (strID)
{
  var n = ScormObjectiveGetIndex(strID);
  if (n > -1) // An objective record exists with this identifier
  {
    return n;
  }
  n = ScormObjectiveGetCount();
  var strPrefix = "cmi.objectives." + n + ".";
  if (ScormSetValue(strPrefix + "id", strID) != "true") return -1;
  return n
}
```

```
function ScormObjectiveGetCount()
{
    var r = parseInt(ScormGetValue("cmi.objectives.count"));
    if (isNaN(r)) r = 0;
    return r;
}
function ScormObjectiveGetData(strID, strElem)
{
    var n = ScormObjectiveGetIndex(strID);
    if (n < 0)
    {
        return ""; // No objectiverecord exists with this identifier
    }
    return ScormGetValue("cmi.objectives." + n + "." + strElem);
}
function ScormObjectiveGetIndex(strID)
{
    var n = ScormObjectiveGetCount();
    for (i = 0; i < n; i++)
    {
        if (ScormGetValue("cmi.objectives." + i + ".id") == strID)
        {
            return i;
        }
    }
    return -1;
}
function ScormObjectiveSetData(strID, strElem, strVal)
{
    var n = ScormObjectiveGetIndex(strID);
    if (n < 0) // If no objective record with this ID
    {
        n = ScormObjectiveAddRecord(strID);
        if (n < 0) return "false"; // No objective record and failed to create one
    }
    return ScormSetValue("cmi.objectives." + n + "." + strElem, strVal);
}
}
```

- ScormObjectiveAddRecord toma como parámetro el identificador del objetivo, y retorna un entero que es el índice del registro del objetivo o -1 si ocurre un error. Si un registro de objetivo con el mismo identificador ya existe, la función no hace nada y retorna el índice del registro existente. Si el número de registros de objetivos permitidos excediera por la adición de un nuevo registro, la función falla.
- ScormObjectiveGetCount no toma como entrada ningún parámetro y retorna un valor entero representando el número de registros objetivos existentes.
- ScormObjectiveGetData toma como parámetro el identificador del objetivo y el elemento del modelo de datos dentro del registro del objetivo, y retorna un valor. Si el valor es una cadena vacía, podría tratarse de un error. La cadena que identifica el elemento de datos es la que aparece a la derecha de "objectives.n.." en la documentación de SCORM para el modelo de datos.
- ScormObjectiveGetIndex toma como parámetro el identificador de un objetivo, y retorna un entero que es el índice del registro de objetivos o bien -1 si no existe tal registro. Esta función puede usarse para saber si un registro ya existe para una interacción.
- ScormObjectiveSetData toma como parámetros el identificador de la interacción, el elemento del modelo de datos dentro del registro de interacción, y el valor a poner. Retorna cierto o falso. Si retorna falso, se puede analizar el estado de error. Dado que los datos no pueden ser almacenados en un registro de interacción hasta que el identificador es almacenado en el registro, la función intentará llamar a ScormObjectiveAddRecord si necesita crear un registro. La cadena que identifica el elemento de datos es la que aparece a la derecha de "objectives.n.." en la documentación de SCORM para el modelo de datos.

Usando esta API se puede crear el siguiente SCO:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>A simple SCO with interaction tracking</title>
<script type="text/javascript" src="ostyn2004sco.js"> </script>
<script type="text/javascript">
function MarkObjectiveSuccess(strID)
{
    ScormObjectiveSetData(strID, "success_status", "true")
    ScormCommit();
}
</script>
</head>
<body> <!-- onload and onunload are managed by ostyn2004sco.js -->
<form id="urn_ostyn_rcdid_sa20041101_bar" onsubmit="MarkObjectiveSuccess(this.id)">
Click the button to mark this objective as successful.
<input type="submit" value="Done" /></form>
</body>
</html>

```

Este SCO devuelve algunos datos del estado acerca de una pareja de objetivos, y esta configurado para que siempre ponga en el estado los mismos valores.

8. FINALIZACIÓN DE LA EJECUCIÓN DE UN SCO

Existen dos formas para que un LMS lance un SCO, en un frame o bien en un ventana popup. Si el SCO es lanzado en una ventana popup, es propietario de la pantalla, y puede ser cerrada con seguridad. Sin embargo si el SCO es lanzado en un frame, no es propietario del contexto del frame, el cual será un frameset o una página web. Este contexto es parte del entorno de ejecución. Si el SCO cierra la ventana del navegador que contiene la parte del entorno de ejecución, esto puede producir la pérdida de la información de traza. Si el frame es parte de la interface principal del LMS, cerrando la ventana efectivamente, dará lugar a una mala experiencia de usuario.

En un escenario de secuenciación, el popup y el cierre de ventanas para cada actividad puede que sea una experiencia discordante, especialmente si las actividades son cortas. Cada vez que el usuario elige continuar con la siguiente actividad, la ventana para una actividad cierra y se abre la ventana para la siguiente actividad. En las implementaciones de los LMS se ha seguido la aproximación en la que el frame "stage" permanece constante y los SCOs son secuenciados, o bien la aproximación de un simple popup. Si el entorno de ejecución crea una ventana popup para correr el SCO y lanza el SCO directamente en esa ventana, el SCO se apropia de la ventana y puede cerrarla con seguridad cuando finaliza. De hecho, este es el comportamiento esperado. Si el entorno de ejecución lanza el SCO en un frame, el SO no puede cerrar la ventana que contiene ese frame.

Un SCO puede fácilmente determinar si está corriendo en un ventana de popup o no. Puede chequear si la ventana en la que está lanzado tiene un padre. Si lo tiene, esta corriendo en un frame o frameset. Si no lo está, entonces es seguro cerrar la ventana. Solo el diseñador de un SCO sabe cuando un SCO está finalizado. Típicamente, esto corresponde a algún evento como clicar sobre un botón de Exit, o bien el SCO podría tener su propios mecanismos internos que señala cuando se llega al momento final de la sesión de comunicación.

Si el SCO no puede cerrar su ventana pero la actividad está finalizada, el SCO debería prevenir adicionales interacciones por el usuario con el contenido. El camino más efectivo para hacer esto es seleccionar una vista neutral que sea visualmente armoniosa con el resto de los SCOs pero que solo contenga un ventana emergente que informe al usuario que la actividad ha finalizado, y posteriormente se cierre la sesión de comunicación.

Existen un par de situaciones donde el hecho de que una llamada a un script cierra una ventana, no es una buena idea, incluso si está permitido. El siguiente fragmento de código se evita este problema, para lo cual se ha modificado la función ScormTerminate para intentar cerrar automáticamente la ventana si está permitido. Si la interface de usuario del SCO contienen un botón Exit o un control similar, ese control puede mostrar una interface de usuario neutral y llamar inmediatamente a ScormTerminate() para intentar cerrar la

agrega ⁺

ventana si está permitido. La modificación en el script requiere añadir algunas variables para evitar estados indeseables.

Observar:

- El temporizador para activar el intento de cerrar la ventana comienza antes de llamar a Terminate ().
- El temporizador es una solución a un problema que se da en algunas implementaciones donde la función Terminate() da como resultado un estado extraño, debido a algún error en el script o de la implementación. Aunque el SCO no tiene control sobre como puede estar implementado el LMS, sí puede actuar preventivamente. El temporizador es recurrente, pues los temporizadores de un solo disparo a veces fallan debido al sistema de carga u otros problemas.

```
var gbEnableAutoCloseWindow = true; // Preset value to control the behavior
var gTimerOwnWindowClose = null;
var gbAlreadyTriedToCloseOwnWindow = false;

function ScormTerminate()
{
    var result = "false";
    PrepareCloseWindowIfAllowed();
    if (gnScormSessionState == 1)
    {
        if (gAPI.Terminate("") == "true")
        {
            result = "true";
        }
    }
    return "false";
}

function IsClosingWindowOK()
{
    if (!gbEnableAutoCloseWindow) return false;
    // Tweaking of the rule may be required for some LMS
    // that use what looks like a popup window but is actually a frameset.
    // A function to try to detect such a situation might be inserted here.
    return (window == window.top);
}

function PrepareCloseWindowIfAllowed()
{
    if ((!gbAlreadyTriedToCloseOwnWindow) && (IsClosingWindowOK())
        && (gnScormSessionState == 2))
    {
        gTimerWindowClose = setInterval('SCOCloseOwnWindow()', 1500);
    }
}

function SCOCloseOwnWindow()
{
    if (gTimerOwnWindowClose)
    {
        clearInterval(gTimerOwnWindowClose);
        gTimerOwnWindowClose = null;
    }
    if (gbAlreadyTriedToCloseOwnWindow) return;
    gbAlreadyTriedToCloseOwnWindow = true;
    if (!window.closed) window.close();
}
```

9. OBSERVACIONES

9.1. CONVERSIÓN DE UN ASSET EN UN SCO.

La forma más fácil de convertir un asset en un SCO es embeberlo dentro de una página HTML. Algunas herramientas de autor, permiten hacerlo de manera semiautomática, a través de un método de publicación de plantillas. Para otros assets, se crea una página que contenga un lugar para cargar el asset. Muchos assets, tales como archivos pdf, imágenes, o archivos de texto pueden ser lanzados directamente por un browser. Sin embargo para usarlos como SCO se requieren scripts.

Un SCO genérico puede ser usado como envoltorio para cualquier asset que pueda generarse como un frameset. El script del frameset gestiona su comportamiento como un SCO, y el frame "stage" es usado para mostrar el asset pasivo. El único aspecto particularizable del envoltorio es la URL relativa del asset pasivo, la cual debe ser proporcionada como fuente para el frame "stage". El siguiente ejemplo muestra un SCO que sirve de envoltorio para una imagen JPEG, denominada "foo.jpg":

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>A simple SCO wrapping a passive asset</title>
<script type="text/javascript" src="ostyn2004sco.js"></script>
</head>
<frameset rows="100%,*">
<frame id="StageFrame" src="foo.jpg" /> // here replace src value with URL of asset
<frame id="DummyFrame" src="about:blank" />
</frameset>
</html>
```

Este ejemplo usa comportamientos por defecto construidos dentro del script genérico para reportar los siguientes datos al entorno de ejecución:

- Estado de completitud a incompleto cuando se muestra el asset.
- Estado de completitud a completo cuando el SCO es descargado.
- El tiempo de session, que es el tiempo empleado durante la sesión de comunicación del SCO con el entorno de ejecución. Esto corresponde aproximadamente al tiempo de visualización del asset. Para algunos tipos de asset, tales como un archivo pdf esto incluye el tiempo de carga de la aplicación de ayuda del browser y los datos del asset.

9.2. USANDO FLASH PARA HACER UN SCO.

- Si se usa Flash para crear un SCO, la película Flash debe ser mostrada en un objeto Flash embebido en una página web. La página web en si misma es el SCO, y el elemento resource que describe el SCO en el manifest del paquete SCORM especifica la URL de la página HTML. Para hacer esto se puede usar alguna herramienta de publicación de plantillas Flash.
- No se puede secuenciar usando Flash. Los LMS proporcionan la secuenciación, no el propio contenido. Sin embargo se puede usar Flash para crear cualquier secuenciación dentro de un SCO. Existen muchas formas de llevarlo a cabo, pero una posible consiste en disponer de un único envoltorio Flash maestro que carga diferentes archivos swf dentro del objeto Flash embebido en la página.
- Es altamente recomendado usar JavaScript en la página HTML host que maneja la gestión de las sesiones de comunicación, debido a que los problemas temporales son difíciles de tratar en ActionScript, mientras que las funciones JavaScript pueden responder inmediatamente y síncronamente a los eventos de carga y descarga de eventos. También es recomendable que todas las comunicaciones entre ActionScript y la implementación de la API sean realizadas a través de funciones JavaScript descansando en la página HTML host. Es importante enviar los datos si algo significativo está ocurriendo, en vez de esperar a que finalice la película, ya que esto aumenta la credibilidad, y el final de la película podría nunca llegar a ser alcanzado si el SCO es descargado por un evento de secuenciación.