SCORM-compatible templates

Short instruction on creating SCORM-compatible templates. The given instruction presents a short description of the issues that should be considered by the developer when developing the SCORM-compatible template. Before all, there are several ways of calling SCORM-functions. scormdriver.js driver usage is the most available (and also free) method. Although the driver is out-of-date, it still works and might be used. Further instruction concerns working with scormdriver.js functions.

Directory structure

Every template must have the following structure:

```
common
extend
scormcontent
    static
    index.html
scormdriver
unique
vocab
```

Such a structure is achieved by uploading the LUCY template to SCORM. All the files connected with the template are located in scormcontent folder. The same folder contains a static folder where all necessary files (e.g. scripts or images) are located. Also, the static folder might as well contain various subfolders. The SCORM root folder contains imsmanifest.xml that allows defining templates names. For example:

```
<?xml version="1.0" standalone="no" ?>
<manifest identifier="RusticiSoftwareSCORMDriverTemplate" version="1.3"
xmlns="http://www.imsglobal.org/xsd/imscp_v1p1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:adlcp="http://www.adlnet.org/xsd/adlcp_v1p3"
xmlns:adlseq="http://www.adlnet.org/xsd/adlseq_v1p3"
xmlns:adlnav="http://www.adlnet.org/xsd/adlnav_v1p3"
xmlns:imsss="http://www.imsglobal.org/xsd/imsss"
xsi:schemaLocation="http://www.imsglobal.org/xsd/imscp_v1p1 imscp_v1p1.xsd
                  http://www.adlnet.org/xsd/adlcp_v1p3 adlcp_v1p3.xsd
                  http://www.adlnet.org/xsd/adlseq_v1p3 adlseq_v1p3.xsd
                  http://www.adlnet.org/xsd/adlnav_v1p3 adlnav_v1p3.xsd
                  http://www.imsglobal.org/xsd/imsss imsss_v1p0.xsd">
<metadata>
  <schema>ADL SCORM</schema>
  <schemaversion>2004 4th Edition</schemaversion>
</metadata>
<organizations default="B0">
<organization identifier="B0" adlseq:objectivesGlobalToSystem="false">
<!--****** Title ******-->
<title>Name of your template</title>
```

```
<item identifier="i1" identifierref="r1" isvisible="true">
<!--****** Title ******-->
<title>Name of your template</title>
              ...
              ...
              ...
```

The <title> tag should contain the template's name in both parts of the document.

index.html file structure

There is only one particularity about the file - file path. In LUCY path has the following structure: %static%/file. However. in SCORM all the paths should begin with ./static/... The template may contain any amount of HTML-files. Switching between files may as well be performed with plain links (e. g. <a href="file.html"...) and by means of uploading files within AJAX. Thus there are no major differences between LUCY- and SCORM-templates in this respect.

SCORM functions

One has to enable the driver in order to use SCORM functions. The driver is enabled through the /imsmanifest.xml file:

```
     ...
<resources>
<resource identifier="r1" type="webcontent" adlcp:scormType="sco"
href="scormdriver/indexAPI.html">
<file href="scormdriver/indexAPI.html" />
              ...
<file href="scormdriver/scormdriver.js" />
```

On the next step the driver is being initialized. Here is a rough example:

```
var SCORMDriver;
var fSCORM = false;
$(function() {
    SCORMDriver = window.parent;
        if(typeof SCORMDriver.SetScore === 'undefined') {
            console.log("no SCORM", SCORMDriver);
        }
        else {
            console.log("SCORM");
            fSCORM = true;
        }
});
```

fSCORM variable is set as TRUE if the template is launched through SCORM, otherwise the variable is set as FALSE. Thus, one can create a template that operates both through SCORM and LUCY. The following functions are available for the template developer: The function below returns the Score

variable value (it contains the result of course completion).

```
function lucyGetScoreScorm() {
    if(fSCORM) {
        var score = SCORMDriver.GetScore();
        if(score == "") {
            return 0; /* course is launched for the first time */
        }
        return score; /* in case the course has already been launched the
previous result is returned */
    }
    return 0; /* the course is not launched through SCORM */
}
```

This function stores the Score value. In this particular example the template contains a course that consists of 10 questions: var lucyQuizQuestions = 10; If user answers 4 questions as an example, lucySetScoreScorm(4) function is called. in this case the Score is calculated: score = 4 * 100 / 10 = 40 %

```
function lucySetScoreScorm(nAnswers) {
    var score = nAnswers*100/lucyQuizQuestions;
    if(fSCORM) {
        if(score > 99) {
            SCORMDriver.SetScore(100, 100, 0); /* all the questions are
answered, result = 100% */
            SCORMDriver.SetPassed(); /* the course is considered passed */
        }
        else {
            SCORMDriver.SetScore(score, 100, 0); /* transfering the result
to SCORM */
        }
        /* CommitData() function must be called in order to avoid data loss
in case user closes the window unexpectedly */
        SCORMDriver.CommitData();
    }
}
```

The following function might be called in case one needs to set “Passed” status:

```
function endCourse() {
    console.log("End of Course");
    if(fSCORM) {
        SCORMDriver.SetPassed();
        SCORMDriver.CommitData();
    }
}
```

Testing

Free service https://cloud.scorm.com is recommended in terms of testing the template. Although it has templates size restrictions, any template might be debugged. Since the service has a friendly user interface it is rather easy to work with. However, one should remember that as a rule clients use custom shells, therefore, certain differences of the SCORM templates actions might occur. One should always keep in mind that the SCORMDriver.CommitData() function should be necessarily called every time the result is uploaded to SCORM.

Languages

Unfortunately there no shortcuts for creating multilingual templates. In order to create the template in any new language, one has to copy the whole directory structure creating a separate template as a result. Thus, if the template should include three languages one has to create three different templates (each for the particular language).

LUCY- and SCORM-compatible templates creation

First of all, the variables that store current state should be gathered in one particular place of the templates structure. For example: At the bottom of the index.html one enables the following script:

```
<script type="text/javascript" src="./static/js/ext.js"></script>
```

The script itself:

```
$(function() {
    var sChunk = ""; //the template is launched within SCORM check
    SCORMDriver = window.parent;
    if(typeof SCORMDriver.SetScore === 'undefined') {
        //This is not SCORM
        console.log("no SCORM", SCORMDriver);
    }
    else {
        //Yes, it is SCORM
        console.log("SCORM");
        fSCORM = true;
        //previous state check
        //current state is being saved periodically so one could
        //check the state when the script is started.
        sChunk = SCORMDriver.GetDataChunk();
        console.log("chunk:",sChunk);
    }
    //in case the state is not null one transforms the string
    //to javascript variables
    //in case the current state is null one saves a word “empty”
    //in order to be able to see it in  $('div.debug-scorm') debug window
    //within the first launch there is no current state
    //thus SCORMDriver.GetDataChunk() returns an empty string.
    if(sChunk)
        Course.oState = JSON.parse(sChunk);
```

```
    else
        sChunk = "empty";
     console.log("loadState",Course.oState);
     sChunk = sChunk.replace(/,/g,",<br>");
     //current state is transmitted to debug window. It is not vital in
terms of the templates operation
     //but it is convenient in terms of debugging
     $('div.debug-scorm').html(sChunk);
});


//save state
//current templates state is being saved
function saveState() {
     //transforming current state to string
     var s = JSON.stringify(Course.oState);
     //Course.oState.allPoints variable stores general result of course
completion
     //(all the quizzes results )
     //in this particular template user is able to reach maximum of 600
points
     //transforming points to percents. 600 points equals 100%
     var score = Course.oState.allPoints * 100/600;
     if(fSCORM)
     {
          //current state is saved to SCORM
          SCORMDriver.SetDataChunk(s);
          //600 points --> 100%
          //the result is saved in percents.
          //100% - maximum (The second parameter states that)
          SCORMDriver.SetScore(score, 100, 0);
          //400 points --> passed
          //lower threshold is 400 points for this particular course
          if(Course.oState.allPoints >= 400)
             SCORMDriver.SetPassed();
          //the function is called in order to avoid data loss
          SCORMDriver.CommitData();
     }
     console.log("saveState", s, score);
}


// the following function is called in case the user passes through the
whole course to its end

function endCourse() {
    console.log("End of Course");
    if(fSCORM) {
        SCORMDriver.SetPassed();
        //SCORMDriver.SetReachedEnd();
        SCORMDriver.CommitData();
```

```
    }
}
```

More of the code that is added to the template itself:

```
var Course = {
    //this object stores current state of the template
    oState: {
        currLesson: 0, //current lesson number
        currFragment: 0, //current fragment number (in case the template
includes several fragments)
        avatar: -1, //0 - Sam, 1 - Sue avatar: either female (Sue) or male
(Sam) are available to choose
        //the course is divided into so-called cards. Each lesson contains
several cards (sublessons)
        //when another card is passed a corresponding flag is marked
        f_card1: 0, //Klassifizierung   bits: 0...4 - topic1...topic5 of the
card1 are shown
        f_card2: 0, //Sichere E-Mail Kommunikation
        f_card3: 0, //
        f_card4: 0, //Bankkundendaten bei SIX  lesson16 (bits: 0...2 -
topic1...topic3)
        f_card21: 0, //Unterlagen und mobile Gerate
        f_card22: 0, //Gesprache in der Offentlichkeit
        f_card23: 0, //Nutzung privater oder offentlicher Gerate
        f_card24: 0, //
        f_card31: 0, //Sicheres Passwort
        f_card32: 0, //Clean Desk und Clear Screen
        f_card33: 0, //Badge
        f_card34: 0, //
        f_card41: 0, //Fluchtwege
        f_card42: 0, //Brandfall
        f_card43: 0, //
        nCards: 0, //collected cards
        nPoints: 0.0, //points
        allPoints: 0.0, //all points (cards and exams)
        //The course includes 4 challenges. Each challenge gives up to 100
points.
        //following variables contain points for each challenge
        nPoints11: 0.0,     //Challenge1
        nPoints26: 0.0,     //Challenge2
        nPoints31: 0.0,     //Challenge3
        nPoints41: 0.0,        //Challenge4
        //main menu allows to reach a particular lesson
        //at the very beginning of the course all the menu options are of
restricted access
        //as soon as the lesson is passed the corresponding option becomes
available
        fEnaLesson6: false, //true - enable menu "Lesson1"
        fEnaLesson20: false, //true - enable menu "Lesson2"
        fEnaLesson30: false, //true - enable menu "Lesson3"
```

```
        fEnaLesson40: false, //true - enable menu "Lesson4"
        fEnaLesson50: false //true - enable menu "Lesson5"
    },
        ..............
};
```

As one is able to see the Course.oState object stores the current state of the template. Depending on the particular template this object may contain different variables. One should keep in mind that Course.oState object has to be necessarily created when a new template is developed. Older courses store state within Lesson71.percent, Lesson72.percent, and etc. variables. Newer ones should have these variables stored inside of Course.oState