eXeLearning- Important information and Setup Guide

1.0 Additional Setup Instructions

To ensure the new iDevices and SCORM functions work properly:

1. **Install the Correct Version of eXeLearning:**
   - Download and install the full *install version* of eXeLearning (not portable or ready to run versions).
   - This ensures the application registers correctly with system paths and dependencies.
2. **Administrator Access:**
   - You must have **administrator rights** on the computer to edit or add files to protected directories like C:\Program Files (x86)\exe\scripts\idevices.
3. **After Updating or Adding iDevices:**
   - Clear the eXeLearning application cache stored in: APPDATA\exe
   - Restart eXeLearning. This ensures the program loads the updated or newly added iDevices correctly.

2.0 Source Code Modifications

File: SCOFunctions.js

Directory: In scripts directory within eXeLearning installation path (e.g., C:\Program Files (x86)\exe\scripts)

Instruction:

1. Replace the function unloadPage(isSCORM) with the following
2. add the finish course function
3. Ensure goBack() and goForward() functions are included.

```
// Mark the course as completed and successful
function finishCourse() {
  computeTime();

  if (typeof pipwerks !== "undefined" && pipwerks.SCORM) {
      // Mark this SCORM package as completed and passed
      pipwerks.SCORM.SetCompletionStatus("completed");
      pipwerks.SCORM.SetSuccessStatus("passed");

      // Save and quit SCORM session
      pipwerks.SCORM.save();
      pipwerks.SCORM.quit();
  } else {
      console.warn("SCORM API not available. Unable to set completion
status.");
  }
}

function unloadPage(isSCORM) {
  if (typeof isSCORM === "undefined") {
      isSCORM = false;
  }

  if (exitPageStatus !== true) {
      if (scorm.GetCompletionStatus() !== "completed") {
          scorm.SetCompletionStatus("incomplete"); // Ensure incomplete if not
finished
          scorm.SetSuccessStatus("failed");
      }
      doQuit();
  }
}

function goBack() {
  pipwerks.nav.goBack();
}

function goForward() {
  pipwerks.nav.goForward();
}
```

3.0 Original code from SCOFunctions

```
function unloadPage(isSCORM)
{

    if (parent && !parent.mod_scorm_is_window_closing){
        // #505 Issue
        parent.mod_scorm_is_window_closing = true
    }

    if (typeof isSCORM == "undefined"){
        isSCORM = false;
    }
    //console.trace('exitPageStatus'+exitPageStatus);

    var status;
    if (exitPageStatus != true)
    {
        status = scorm.GetSuccessStatus();
        // In SCORM12, information about completion and success is stored in
the same place (cmi.core.lesson_status)
        if (status!="passed" && status!="failed")
        {
            if(isSCORM==true)
            {
                scorm.SetCompletionStatus("incomplete");
                scorm.SetSuccessStatus("failed")
            }
            else
            {
                scorm.SetCompletionStatus("completed");
                scorm.SetSuccessStatus("passed")
            }
        }
        doQuit();
    }

    // NOTE:  don't return anything that resembles a javascript
    //        string from this function or IE will take the
    //        liberty of displaying a confirm message box.
}
```

4.0 New iDevices

How to Create an Idevice

1. **Duplicate an Example Idevice**:
   - To simplify the setup, copy an existing idevice folder (e.g., "example-idevice") and rename it.
   - Ensure that the new folder structure includes config.xml, edition, and export.
2. **Edit config.xml**:
   - Open config.xml and update the idevice's name, description, and other identifiers to reflect the new idevice's purpose.
   - This customization allows eXeLearning to recognize the idevice as a unique option.
3. **Develop JavaScript/CSS in the edition and export Folders**:
   - Customize the JavaScript in the edition folder for editing functionality and in the export folder for published interactivity.
   - Include any CSS required for specific styling.
4. **Restart eXeLearning and clear AppData/exe**:
   - After creating and configuring the idevice, clear your cache in AppData/exe and restart eXeLearning to load the new idevice. It should now appear as an option in the idevice selection menu.

Folder Structure and Key Components of iDevices

1. **Config File (config.xml)**
    - This file is essential for defining the idevice's basic configuration. It contains metadata and settings such as the idevice's name, description, and icon.
    - The config.xml file also includes the structure that eXeLearning uses to recognize and display the idevice in its editor, allowing it to appear as a selectable option within eXeLearning's user interface.
2. **Edition Folder**
    - The edition folder holds the JavaScript (and CSS if needed) that controls the behavior and appearance of the idevice during the editing phase in eXeLearning.
    - The JavaScript file in this folder is responsible for generating the form fields or interactive elements that appear when an LMS author edits the idevice's content in eXeLearning.
    - **Example**: If the idevice includes a text input or a button, the JavaScript in the edition folder would define these elements and any custom functionality (e.g., character limits, validation, or dynamic responses).
3. **Export Folder**
    - The export folder contains the JavaScript and CSS files necessary for the idevice's functionality and styling when it is published to SCORM or exported from eXeLearning.
    - This folder ensures that the interactive features, such as buttons or progress tracking functions, work correctly on the LMS once the content is deployed.
    - **Example**: If the idevice includes a "Next" button that tracks completion, the JavaScript in the export folder would handle the completion status update and any additional interactions required for SCORM functionality.

5.0 Tabular View SCOFunctions Changes

Practical Impact of the Changes

Feature	Original Code	First Modification	Second Modification (Latest)	Benefit
Completion Tracking	Relied on scorm.GetSuccessStatus(), which did not always update completion accurately. **Code:** js status = scorm.GetSuccessStatus(); if (status!="passed" && status!="failed") { scorm.SetCompletionStatus("incomplete"); scorm.SetSuccessStatus("failed"); }	Used scorm.GetCompletionStatus() !== "completed" to ensure accurate status updates. **Code:** js if (scorm.GetCompletionStatus() !== "completed") { scorm.SetCompletionStatus("incomplete"); scorm.SetSuccessStatus("failed"); }	No changes in this section since first modification. **Code remains:** js if (scorm.GetCompletionStatus() !== "completed") { scorm.SetCompletionStatus("incomplete"); scorm.SetSuccessStatus("failed"); }	Prevents courses from getting stuck in an **incomplete** state.
Success Status Handling	Used a **fixed logic** where SCORM packages were either "completed" or "failed" based on a simple check. **Code:** js if(isSCORM==true) { scorm.SetCompletionStatus("incomplete"); scorm.SetSuccessStatus("failed"); } else { scorm.SetCompletionStatus("completed"); scorm.SetSuccessStatus("passed"); }	Differentiated between **package completion** and **course completion** using an explicit function. **Code:** js if (isFinalPackage) { pipwerks.SCORM.SetSuccessStatus("passed"); } else { pipwerks.SCORM.SetSuccessStatus("incomplete"); }	Now **always** marks completion as "passed" without needing isFinalPackage. **Code:** js pipwerks.SCORM.SetSuccessStatus("passed");	Avoids incorrect "passed" status for partially completed courses.
Handling LMS Communication Issues	No explicit handling for SCORM API failures, causing **silent errors** when SCORM functionality was unavailable.	Added a **SCORM API availability check** to prevent errors and notify the console when the API is missing. **Code:** js if (typeof pipwerks !== "undefined" && pipwerks.SCORM) {	No changes in this section since first modification. **Code remains:** js if (typeof pipwerks !== "undefined" && pipwerks.SCORM) { pipwerks.SCORM.SetCompletionStatus("compl	Ensures SCORM completion is **only attempted when API**

		pipwerks.SCORM.SetCompletionStatus("completed"); pipwerks.SCORM.save(); pipwerks.SCORM.quit(); } else { console.warn("SCORM API not available. Unable to set completion status."); }	eted"); pipwerks.SCORM.SetSuccessStatus("passed"); pipwerks.SCORM.save(); pipwerks.SCORM.quit(); } else { console.warn("SCORM API not available. Unable to set completion status."); }	**is available**, preventing silent failures.
User Feedback (Alerts)	**No alerts for completion**, making it unclear when the SCORM package was successfully completed.	**Added alerts** when the SCORM package or entire course is completed. **Code:** js if (isFinalPackage) { alert("Course completed!"); } else { alert("SCORM package completed!"); }	**Alerts removed** - now, SCORM completion happens **silently** without pop-ups. **Code:** js pipwerks.SCORM.SetCompletionStatus("completed"); pipwerks.SCORM.SetSuccessStatus("passed"); pipwerks.SCORM.save(); pipwerks.SCORM.quit();	Eliminates **unnecessary alerts**, allowing SCORM to complete **without user interruption**.
Unload Page Behavior	Used scorm.GetSuccessStatus() to determine if SCORM should be marked **incomplete or completed** upon unloading.	Ensured that when a page is **unloaded**, it correctly updates SCORM status. **Code:** js if (scorm.GetCompletionStatus() !== "completed") { scorm.SetCompletionStatus("incomplete"); scorm.SetSuccessStatus("failed"); } doQuit();	No changes in this section since first modification. **Code remains:** js if (scorm.GetCompletionStatus() !== "completed") { scorm.SetCompletionStatus("incomplete"); scorm.SetSuccessStatus("failed"); } doQuit();	Prevents SCORM sessions from being incorrectly **marked as failed or incomplete** when exiting.
Navigation Enhancements	No navigation functions included.	No navigation functions included.	**Added goBack() and goForward() functions for SCORM navigation.** **Code:** js function goBack() { pipwerks.nav.goBack(); } function goForward() { pipwerks.nav.goForward(); }	Allows users to **navigate SCORM content smoothly** between pages.