Custom Multimedia Lesson

Documentation

Based on Version A0.0.1

Table of Contents

Disclaimer

Most of this documentation was created by AI based on the source code submitted to it. This document has been edited and checked for accuracy. It is accurate as of 11/22/2025

Course Player Controller (script.js)

Overview

script.js is the core logic file for the index.html parent window. It manages the application state, handles navigation between course pages, tracks user progress (video completion, scrolling, quiz scores), and persists data to localStorage.
It functions as a lightweight SCORM wrapper, communicating with content loaded inside an iframe via the window.postMessage API.

Data Structure

The application state is encapsulated entirely within the state object. The primary data model is stored in state.data.

State Properties

Property	Type	Description
pages	Array	Loaded dynamically from course_data.json. Contains logic rules, watch time, scores, and completion status for each page.
currentPageIndex	Integer	The array index of the currently visible page.
progress	Integer	The furthest index the user has successfully completed.
totalCourseSeconds	Integer	Total time spent in the course (updated every second).
log	Array<String>	debug/audit log of user actions and system events.

Message API (Parent <-> Child Communication)

Communication between the Parent (index.html) and the Child (iframe content) is handled via the window.postMessage API. The state.handleMessage function processes incoming signals.

Incoming Messages (Child to Parent)

The Child (iframe) sends these messages to the Parent to update the state.

Type	Payload (message)	Description
LOG	String	Adds an entry to the central system log (e.g., "User clicked play").
PAGE_SCROLLED	Boolean (via scrolled key)	Sent when the user scrolls to the bottom of a text page. Updates page.scrolled.
VIDEO_PROGRESS	Float (0.0 - 1.0)	Updates the video completion percentage for the current page.
QUIZ_ADD_QUESTIONS	null (or ignored)	Triggers the Parent to render quiz HTML based on JSON data and send it back to the child.
QUIZ_SUBMITTED	Object (Responses)	Contains user answers. Triggers gradeQuizQuestions, updates the score, and attempts to finalize the page.

Outgoing Messages (Parent to Child)

The Parent sends these messages to the lessonFrame content window to manipulate the UI or return data.

Type	Payload	Description
QUIZ_ADD_QUESTIONS	HTML String	The rendered HTML form for the quiz. Sent in response to an incoming QUIZ_ADD_QUESTIONS request.

QUIZ_INFO	{ score: Float, maxAttempts: Int }	Sent after a quiz is submitted and graded. Tells the child page the result so it can show feedback.

Core Methods

Initialization & Navigation

- **init(frameId, bannerId)**:
    - Captures DOM elements for the iframe and info banner.
    - Starts the 1-second interval timer which increments watchTime for the specific page and totalCourseSeconds for the global state.
    - Auto-saves every 60 seconds.
- **loadCourseData()**: Asynchronously fetches course_data.json. It initializes new properties for every page (score, attempts, etc.) that aren't present in the raw JSON.
- **next()**:
    - Checks finalizePage() first.
    - If page.completed is true, increments the index and loads the new page.
    - If the page is not complete, displays a banner error.
- **prev()**: Decrements the index and loads the previous page. Prevents going below index 0.

Logic & Grading

- **checkIfComplete()**: Returns true only if **all** criteria in the course_data.json completion rules are met:
    1. watchTime >= required time.
    2. score >= required score.
    3. scrolled match required boolean.
    4. videoProgress >= required percentage.
- **finalizePage()**: Calls checkIfComplete(). If the page is done, it marks page.completed = true, increments global progress, and triggers a save.
- **gradeQuizQuestions(questions, responses)**:
    - Comparison Logic: Converts both the Correct Answer and User Response to **Lower Case** and **Sorts** them. This ensures case-insensitivity and order-independence (for multi-select questions).
    - Calculates a score (0.0 to 1.0) based on point values defined in the JSON.

- **renderQuiz(index)**: Generates raw HTML strings for the quiz form. This is done on the parent side to keep the content JSON secure and separated from the display logic.

Persistence

- **save()**: Serializes state.data to a JSON string and saves it to the browser's localStorage under the key "courseProgress".
- **loadSave()**: Checks localStorage. If found, it uses Object.assign to merge the saved user data with the structure loaded from course_data.json.
- **reset()**: Wipes the localStorage entry, reloads the window, and effectively restarts the course.

Files System Dependencies

- **course_data.json**: This file must exist relative to index.html. It contains the array of page objects, questions, and completion rules.
- **script.js**: This file.
- **index.html**: Must contain an iframe with ID lesson-frame and a div with ID info-banner (or IDs passed to .init).

Course Data Schema (course_data.json)

Overview

The course_data.json file acts as the "Manifest" or "Playlist" for the course. It defines the order of pages, the type of content on each page, the logic required to "pass" that page, and the data needed to render dynamic content (like quizzes).
The application loads this file into state.data.pages on startup.

Root Structure

The file must be a single **JSON Array** containing **Page Objects**. The order of objects in the array determines the navigation order (Next/Prev) in the course.

JSON

```
[
  { "type": "article", ... },
  { "type": "quiz", ... },
  { "type": "video", ... }
]
```

Page Object Schema

Every page object requires standard metadata and a set of completion rules.

1. Common Properties

These properties apply to all page types.

Property	**Type**	**Description**
type	String	Identifies the content type. Common values: "article",

		"quiz", "video".
name	String	The filename (relative path) of the HTML file to load in the iframe (e.g., "page1.html").
completionRules	Object	**Required.** Defines the strict criteria to mark the page as "Completed".

2. Completion Rules Object

The completionRules object acts as a gatekeeper. The script.js controller checks these rules every second. **ALL** conditions must be met for the page to complete.

Property	**Type**	**Logic Applied**
watchTime	Integer	**Minimum Seconds.** The user must stay on this page for at least this many seconds. Set to 0 to ignore.
score	Float	**Minimum Score.** (0.0 to 1.0). Used for quizzes. 0.7 = 70%. Set to 0.0 for non-graded pages.
scrolled	Boolean	**Scroll Requirement.** If true, the user receives a "PAGE_SCROLLED" event from the child page. If false, this rule passes automatically.
videoProgress	Float	**Video Completion.** (0.0 to 1.0). If set to 0.95, the user must watch 95% of the video. Set to 0.0 if no video exists.
attempts	Integer	**Max Attempts.** Used specifically for Quizzes to limit retries. Passed to the child page via QUIZ_INFO.

3. Quiz-Specific Properties

If type is "quiz", the object must include a questions array.

Question Object

Property	Type	Description
id	String	Unique identifier for the question (e.g., "Q1").
text	String	The actual question text displayed to the user.
possibleAnswers	Array<String>	List of all options displayed as checkboxes/radios.
correctAnswers	Array<String>	List of strings that **must** match the possibleAnswers. Case-insensitive.
pointValue	Integer	How much this question contributes to the total score.

Example: Creating a Short Quiz

To add a generic "Knowledge Check" quiz to your course, append a new object to your main JSON array.

The Scenario

You want a quiz stored in a file called quiz_template.html. It should have:

- **Two questions.**
- **Passing Score:** 100% (1.0).
- **Time Limit:** User must spend at least 10 seconds on the page.
- **Max Attempts:** 2 tries allowed.

The JSON Snippet

Copy and paste this object into your course_data.json array.

JSON

```
{
  "type": "quiz",
  "name": "quiz_template.html",
  "completionRules": {
    "watchTime": 10,
    "score": 1.0,
    "scrolled": false,
    "attempts": 2,
    "videoProgress": 0.0
  },
  "questions": [
    {
      "id": "Q1",
      "text": "Which file handles the course logic?",
      "correctAnswers": ["script.js"],
      "possibleAnswers": ["index.html", "script.js", "style.css"],
      "pointValue": 5
    },
    {
      "id": "Q2",
      "text": "Select all valid page types:",
      "correctAnswers": ["quiz", "video"],
      "possibleAnswers": ["quiz", "banana", "video", "car"],
      "pointValue": 5
    }
  ]
}
```

How the Grading Works (Based on script.js)

- **Case Insensitive:** If the user selects "Script.js" (Capitalized) but your JSON says "script.js" (lowercase), the system **will** mark it correct.
- **Order Insensitive:** In Q2, if the user checks "video" then "quiz", but your JSON lists ["quiz", "video"], it **will** mark it correct.
- **Strict Matching:** The user must select **ALL** correct answers and **NO** incorrect answers to get the points for that question.