Custom Multimedia Lesson

Documentation

Based on Version A0.2.0

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 11/29/2025

Technical Documentation: Custom Multimedia Lesson (CML)

Overview

This application is a vanilla JavaScript "Single Page Application" (SPA) designed to run inside a standard Learning Management System (LMS) via SCORM 1.2 or as a standalone web app. It separates the **Business Logic** (State), **LMS Connectivity** (LMS Adapter), and **Data Encoding** (Journaler) into distinct modules.

Architecture

The application follows a **Controller-Adapter-Utility** pattern:

- **script.js (Controller):** Manages the UI, user flow, and business rules (e.g., "Can the user go to the next page?").
- **lms.js (Adapter):** Handles data persistence. It abstracts the SCORM API, providing a unified interface for saving/loading data regardless of whether the app is online (SCORM) or offline (LocalStorage).
- **journaler.js (Utility):** Handles data serialization, compression, and logging. It turns complex JavaScript objects into efficient strings for storage.

1. Data Schema (The "Delta")

To maximize the 4,096 character limit of SCORM 1.2, the application does **not** save the full state object. Instead, it saves a compressed array of values representing only the user's progress.

The Serialization Map

The state.data.delta object is flattened into an array of integers and strings before saving. The order of this array is strict.

Index	Variable	Type	Description
0	currentPageIndex	Int	The index of the currently visible page (0 to N).
1	progress	Int	The highest page index the user has unlocked/completed.
2	totalCourseSeconds	Int	Lifetime seconds spent in the course.
3	completed	0 / 1	**[Start of Page Loop]** Status of Page 0.
4	scrolled	0 / 1	Status of Page 0.
5	score	Int	Score of Page 0 (Saved as Int 0-100, converted to Float 0.0-1.0 on load).
6	watchTime	Int	Seconds spent on Page 0.
7	attempts	Int	Number of times the quiz was submitted on Page 0.
8	videoProgress	Int	Furthest video percentage (0-100) on Page 0.
9	userAnswers	String	JSON string of user inputs {"Q1":["A"]}. Sanitized to remove delimiters.
10	completed	0 / 1	**[Start of Page 1 Loop]** ... and so on.

2. Module: LMS Manager (lms.js)

This module implements the **Adapter Pattern**. It detects the environment and loads the appropriate driver.

Core Methods

- **init()**: Auto-detects SCORM API. If found, loads Scorm12Adapter. If not, loads LocalStorageAdapter.
- **saveData(string)**: Commits the compressed string to the persistence layer.
- **loadData()**: Retrieves the string. Returns null if no save exists.
- **setScore(raw, max)**: Reports the grade to the LMS gradebook.
- **setStatus(string)**: Sets the course status ("passed", "failed", "completed", "incomplete").
- **quit()**: Terminates the session (calculates session time in the state before calling this).

Drivers

- **Scorm12Adapter**: Maps calls to cmi.suspend_data, cmi.core.score.raw, etc. Handles error checking (size limits) and transactions (LMSCommit).
- **LocalStorageAdapter**: Mimics the SCORM API using browser localStorage. Used for development and offline playback.

3. Module: Telemetry & Encoding (journaler.js)

This module acts as a "Black Box Recorder." It captures events, timestamps, and compresses data.

Features

- **Base36 Encoding:** Converts large integers (timestamps) to short alphanumeric strings to save space.
- **Gzip Compression:** Uses the browser's CompressionStream API to compress the save string before sending it to the LMS.
- **Event Logging:** Records granular user actions (VIDEO_PLAY, QUIZ_SUBMITTED) into a chronological log buffer.

Public API

- **log(action, value)**: Adds an event to the buffer.
    - *Example:* journaler.log("VIDEO_PLAY", "0,50") -> Stores a2,a,1e (Timestamp, ActionCode, Value).
- **pack(deltaArray)**: Serializes the state array and log buffer, joins them with delimiters (^), and compresses the result.
- **unpack(string)**: Decompresses the data and parses it back into a usable object. Handles fallback for uncompressed legacy data.

4. Module: Application Controller (script.js)

This is the main entry point. It initializes the other modules and manages the DOM.

Initialization Flow

- **window.onload** triggers.
- **lms.init()** establishes connection.
- **state.loadCourseData()** fetches course_data.json and builds the static page objects.
- **state.loadSave()** retrieves data from lms, decompresses via journaler, and hydrates the state.delta object.
- **state.init()** renders the UI (iframe, banner) and starts the timer.

Key Logic

- **finalizePage()**: Checks validity rules (Time, Score, Scroll). If passed, marks page complete and saves.
- **handleMessage(event)**: The "PostMessage API" listener. Receives data from the iframe content (Quizzes, Videos).
    - *Some supported Messages:* QUIZ_SUBMITTED, VIDEO_PROGRESS, PAGE_SCROLLED, LOG.
- **checkCourseCompletion()**: Evaluates global rules (minimumMinutes, minimumGrade) to determine if the user passes the course.

API Reference: Iframe Communication

The parent window (script.js) communicates with the child content (page.html) using window.postMessage.
From Child (Content) -> Parent (Shell):

JavaScript

```
// Example: Reporting video progress
window.parent.postMessage({
    type: 'VIDEO_PROGRESS',
    message: 0.5
}, '*');
```

From Parent (Shell) -> Child (Content):
The shell primarily sends data in response to requests (like rendering a quiz).

JavaScript

```
// Example: Sending quiz HTML
this.lessonFrame.contentWindow.postMessage({
    type: "QUIZ_ADD_QUESTIONS",
    message: "<div>...html...</div>"
}, '*');
```