**API for Game Developers**

**Steps to integrate API into your game:**

1. Include the gamelib.js file in your application.
2. Visit the developer portaland register a developer account.
3. Once logged in you will be provided with a Developer ID.  Keep this page open because you will need this Developer ID when integrating the API into your game.
4. Look at the below API functions and use them where appropriate in your game. Use the following workflow as a guideline:
   1. Call init(callback, developerId) to submit an AJAX request that retrieves the initObj containing data you need for the game (eg. question objects, media URLs, gameId).
   2. Run your game, asking questions to the student in the order that they are provided in the initObj.questions array, starting from the question index specified in initObj.progress.currentAttempt.questionIndex.
      * + Each Question object in the initObj.questions array contains information you need to ask a question, including:
          - text – the question text
          - type – the question type, either multiple choice, short answer, or choose all that apply.
          - possibleAnswers – in the case of multiple choice and all that apply, an array of answers to choose from. In the case of short answer, a singleton array with an object containing the correct answer as text.

The Question object is more fully documented in the next section.

* + - * Questions and possible answers may reference media resources in addition to or instead of text. These references come in the form of id strings. For any given id, initObj.resources[id] retrieves a Resource object containing the URL and MIME type of the resource referenced by that id. URLs can then be used to mark up <img>/<audio>/<video> tags to retrieve the relevant media and display it with the question.
  1. Record the student’s response for each question, and periodically call addStudentAnswers(…) to POST these answers to the database. Use the helper function makeAnswer(...) to construct well-formed Answer objects for submission*. Make sure to post answers according to the question order, and starting from the question index as specified in part b*. You can post multiple times, as long as you keep posting in order.
  2. During your game, you can also call updateGameData(...) to store student-specific game data on the server for your game’s use. The server provides a GameData object for each student-game pair that you may use and format according to your needs. This object is retrieved in initObj.gameData object whenever the game loads, but you can also retrieve it manually at any time by using the getGameData(...) function.

**API**

This section documents the full specification of each API call.

Some of the API functions below take a callback function as a parameter.  The callback function must take a single parameter that is the response from the API function call.

**Example Callback Function:**

function foo(data){

   console.log(data);

}

**init(callback, developerId)**

**Parameters:**

* callback (function(data){}) - function to be executed on response from the server
* developerId (string) - developer specific ID string that can be obtained by logging into your developer account.

This function sets up the assignment to be used throughout the game, and should be called when the game first loads.  For example, you could call this function on window.onload and store the data in the response to be used throughout the game.  Once the API retrieves the relevant data from the server, callback(data) is run.

As a developer, you have access to one test assignment, as well as a single game ID – student ID pair to test the GameData API with (fetchGameData and updateGameData). To access these test data when testing your game, make sure that the browser running the game is logged-in to the TechCaFE server with your developer account. init will then retrieve the test data in the initObj for you.

The specification of the data object fed to callback is as follows.

**Response on Failure:**

data = {

success: false,

error: “Descriptive error message.”

}

**Response on Success:**

data = {

success: true,

initObj: {

assignmentId : '4ddddddddeeeeeeee3000000',

assignmentName : 'Digits',

gameData : <GameData object>,

gameId : '4ddddddddeeeegefd5670000',

isResourceEnabled : true,

minScore : 80,

studentId : '5555dddddeeeegefd5670000',

preferences : {

username : “Hileezy”

},

progress : <AssignmentProgress Object>,

questions: [<Question Object>,

<Question Object>,…], // ORDERED

resources : {

'5ddddddddeeeeeeee3000000' : <Resource Object>,

'5ddddddddeeeeeeee3000001' : <Resource Object>,

...

}

}

}

**Object specifications**

<GameData object> =

{

data : {…}, // You can populate this field with whatever you need

gameId : '4ddddddddeeeegefd5670000',

studentId : '5555dddddeeeegefd5670000'

}

<Question Object> =

{

\_id : "4aaaaaaaaffffffff0000012",

text : “Is this seat \_?” ,

type : "ALL\_THAT\_APPLY",

isResourceEnabled : false,

possibleAnswers : [

{ text : "taken", correct : true},

{ text : "available", correct : true},

{ text : "used", correct : false},

{ text : "sat", correct : false}

]

}, or

{

\_id : "4aaaaaaaaffffffff0000013",

text : "Which one of these is a mug?",

type : 'MULTIPLE\_CHOICE',

isResourceEnabled : true,

possibleAnswers : [

{ resourceId : '5ddddddddeeeeeeee3000000', correct : false},

{ resourceId : '5ddddddddeeeeeeee3000001', correct : true},

]

}, or

{

\_id : "4aaaaaaaaffffffff0000011",

text : "The circled word is spelled incorrectly. What's the correct spelling?",

type : 'SHORT\_ANSWER',

isResourceEnabled : true,

questionResourceId : '5ddddddddeeeeeeee3000002',

possibleAnswers : [

{ text : 'Tip', correct : true}

]

}

To display a question, first determine its type. There are three possible types:

* *Multiple choice* questions have multiple possible answers, one of which is correct. You should thus display all the possible answers and allow the student to choose one.
* *All that apply* questions have multiple possible answers, of which more than one may be correct. You should thus display all the possible answers and allow the student to choose as many he/she thinks is correct.
* *Short answer* questions have one possible answer, and the student must type in that answer to be correct. You should display some sort of text input to capture the student’s response.

Any question may have a questionResourceId in addition to or instead of text, referencing a media resource in the database that should be displayed along with the question. Answers to multiple choice and all that apply questions may also have resourceId’s, and these resources should be displayed with the answer choices.

<Resource Object> =

{

type : ‘AUDIO’ // or ‘VIDEO’ or ‘IMAGE’

mimeType : ‘audio/mp3’ // or "audio/mp4", "audio/mpeg",

// "audio/wav", "audio/vnd.wav”, "image/gif", // "image/jpeg", “image/png", "video/mpeg", // "video/mp4"

url : ‘http://techcafe-server.herokuapp.com/resourcedata/5ddddddddeeeeeeee3000000’

}

To display a resource, use its questionResourceId/resourceId to index into initObj.resources . This will return you a Resource object much like the one shown above, which should provide you with all the information you need. For example, the following HTML can be used to display the audio clip described above:

<audio controls>

<source src= ‘http://techcafe-server.herokuapp.com/resourcedata/5ddddddddeeeeeeee3000000’ type= ‘audio/mp3’>

</audio>

Finally, as long as a question has a question or answer resource, isResourceEnabled will be set to true on the Question object, as a convenience flag.

**makeAnswer(questionId, choice, correct, timeTaken)**

**Parameters:**

* questionId (number) - the id of the question that was answered, which you can retrieve from the Question object (its \_id field).
* choice (object) - the student’s chosen or typed answer(s). The value of this parameter depends on what type of question was asked:
  + *Multiple choice* – choice should be the element the student chose from the Question object’s possibleAnswers array.
  + *All that apply* – choice should be an array containing all the elements the student chose from the Question object’s possibleAnswers array.
  + *Short answer –* choice should be an object { text : … }, where text is set to whatever the student inputted as an answer.
* correct (boolean) - whether or not the answer is correct.
* timeTaken (number) - the time in milliseconds it took to answer the question

Use this function to create an Answer object which you can then post to the server to record the student’s response.

**Return Value:**

* the Answer object formatted correctly and including all of the data that was inputted as parameters to the function.

**addStudentAnswers(answers, timeElapsed, callback)**

**Parameters:**

* answers (Answer[]) - this is an array of Answer objects
* timeElapsed (deprecated; any dummy value will do for this parameter) - the total time playing the game since the assignment was started .
* callback (function(data){}) - the function to be executed on response from the server

This function posts the student’s answers (obtained from makeAnswer(…) as just described) to the server. When posting answers, make sure to follow the ordering constraints described on page 1. If the post is successful, the data response returned by the server will contain an AssignmentProgress object describing all of the student’s progress on this assignment, including past and best attempts, which you may choose to show the student. AssignmentProgress.currentAttempt.questionIndex also gives you the index of the next question (in initObj.questions) to ask.

**Response on Failure:**

data = {

success: false,

error: “Descriptive error message.”

}

**Response on Success:**

data = {

success: true,

assignmentProgress: {

pastAttempts : [

{ score : 5, percentScore : 33.3, dateCompleted :

Date(2014, 4, 1), timeTaken : 15000},

{ score : 10, percentScore : 66.6, dateCompleted :

Date(2014, 4, 2), timeTaken : 14000},

{ score : 9, percentScore : 60, dateCompleted :

Date(2014, 4, 3), timeTaken : 16750},

{ score : 14, percentScore : 94, dateCompleted :

Date(2014, 4, 4), timeTaken : 18000}

],

bestAttempt : { score : 14, percentScore : 94,

dateCompleted : Date(2014, 4, 4), timeTaken : 18000},

currentAttempt : { score : 2, questionIndex : 2, timeElapsed :

1000},

studentId : '4ddddddddeeeeeeee2000005',

assignmentId : '4ddddddddeeeeeeee4000003'

}

}

// (Date(…) represents a Javascript Date object)

**updateGameData(gameData, callback)**

**Parameters:**

* gameData (GameData object) - the GameData object to be saved on the server
* callback (function(data){}) - the function to be executed on response from the server

This function updates the GameData object on the server . The GameData object is defined by the developer, and is specific to a student account for a given game.  You can use this to store information such as high scores, game states, and game specific achievements.

(Note : When you are testing your game (that is, when running your game while logged in as a developer), you will always be given the same gameId in the initObj regardless of what game you are testing. In deployment, however, the gameId will be specific to the particular game being run.)

**Response on Success:**

{

success: true

}

**Response on Failure:**

{

success: false,

error: “Descriptive error message.”

}

**GameData** **Object Structure:**

{

gameId : "4aaaaaaaaffffffff0000013", // Copy from initObj.gameId

studentId : "5ddddddddffffffff0000013", // Copy from initObj.studentId

data : { //game specific data here }

}

**getGameData(gameId, studentId, callback)**

**Parameters:**

* gameId - the id of the game that the developer wants game data for
* studentId - the id of the student that the developer wants game data for
* callback (function(data){}) - the function to be executed on response from the server

This function retrieves a student account’s GameData object for a specific game.

**Response on Success:**

{

success: true,

gameData: { //game data here },

error: “”

}

**Response on Failure:**

{

success: false,

gameData: {},

error: “Descriptive error message.”

}