# JavaScript Library

## Overview

The JavaScript Library is meant to reduce code redundancy when developing projects based in JavaScript. It has a layered architecture that allows for fast, structured, and separated development of each piece. This library contains the following pieces:

* **Ajax Layer** – This is the lowest layer. It takes parameters in a specific interface and then performs the request. If needed, this layer can be extended with a plugin to perform native functions (such as reading in files).
* **RESTful API Definition (RAPID) Layer** – This layer takes in a subset of Ajax Layer parameters and performs formatting and validations for a given method call. This is highly structured so that APIs can be defined through configuration.
* **SDK Interface Layer** – This layer is wrapper around the RESTful API Definition Layer. This layer translates a wrapper interface into the RAPID Layer parameters and can also format the RAPID Layer response. This layer is very free form to create any API interface needed while using the stability of the RAPID Layer. This layer can also be used to perform validations if needed.
* **Extensions/Overrides** – These modifications are in parallel to any other layer. They modify the functionality of a given layer so that additional functionality can be used on a specific platform.

Add a diagram here

Together, the separation of code allows for multiple people to modify multiple pieces at once and allow for building on multiple platforms that all use different variations of JavaScript while optimizing code reuse.

## Build Process

For easy separation of code and project organization, the JavaScript library uses the [require.js](http://requirejs.org), the [require.js optimizer](http://requirejs.org/docs/optimization.html), hasjs integration, and [almond.js](https://github.com/jrburke/almond) for building. The project has the following project structure:

* “buildConfig” folder – Require.js build configuration files
* “builds” folder – current builds for each of the apis
* “lib” folder – library files for external packages
* “src” folder – the source code
* “src/att” folder – main code base and helper functions. This holds code for the Ajax and RAPID Layer code
* “src/extensions” folder – files that add/modify functionality of any component
* “src/wrappers” folder – files that provide new interfaces on top of the RAPID Layer interface

To build a package run the following command from the JavaScript Library folder:

node r.js -o buildConfig/<package-name>.js

To modify the build configuration read more details about the [require.js optimizer parameters](https://github.com/jrburke/r.js/blob/master/build/example.build.js).

## Testing

To prevent minifying during the build process, set the following parameter in the build configuration:

optimization: “none”

Another method is to test generic code in the browser. In the *Plugins/js/test* folder, modify *index.html* to reference the build you want to test by importing it as a script. This allows you to run commands and debug using tools like Firebug and Chrome’s Debug console.

## Ajax Layer

This layer takes in formatted parameters and performs the proper network request. The functionality of this layer can be extended to provide handling of the formatted parameters in the native platform. See *src/att/ajax.js* for a description of the parameter interface.

## RESTful API Definition (RAPID) Layer

This layer is highly structured as a way to format parameters for the Ajax Layer through defined configuration of each of the APIs. For each RESTful API, this configuration can easily and quickly be developed from the documentation.

## SDK Interface Layer

This layer is meant to wrap the RAPID Layer. It is very free form so that it can be used for the following purposes:

* Create an intuitive interface for developers that translate into the RAPID Layer for processing.
* Create an opinionated interface to shield developers from unnecessary complexity.
* Maintain a abstacted interface so that when the response structure from the RAPID Layer change, developers don’t have to modify code to handle the change.
* Provide additional validation that the RAPID Layer doesn’t provide.
* Provide backwards compatibility while utilizing benefits of the RAPID Layer.

## Extensions

Extensions are used to expose or modify functionality for various platforms. This is very helpful because different JavaScript platforms provide different JavaScript implementations of similar features. Extensions replace pieces of the Ajax or the RAPID Layer with pieces that can be used for the specific platform.