A Component Library for jQuery

Introduction

`amplify.request` is an abstraction layer that can be used for any kind of request for data. `amplify.request` sets out to sepearate the data retrieval and caching mechanisms from data requestors.

Using the `amplify.request.define` function you can create and maintain your entire server interface and caching policy in a single place, reducing the impact of any server interface changes. Components that need data can retrieve the data through `amplify.request` without concern for data caching, server interface, location, data types, wrappers, and all the other specificities of the client/server interaction.

Just getting started? Take a look at the examples section below for a few of the popular use cases with `amplify.request`.

Usage

```
amplify.request( string resourceId [, hash data [, function callback ]] )
```

Request a resource.

- `resourceId`: Identifier string for the resource.
- `data`: an object literal of data to be sent to the resource.
- `callback`: a function to call once the resource has been retrieved.

```
amplify.request( hash settings )
```

Request a resource.

- `settings`: a set of key/value pairs of settings for the request.
    - `resourceId`: Identifier string for the resource.
    - `data` (optional): Data associated with the request.
    - `success` (optional): Function to invoke on success.
    - `error` (optional): Function to invoke on error.

```
amplify.request.define(
    string resourceId , string requestType [, hash settings ] )
```

Define a resource.

- `resourceId`: Identifier string for the resource.
- `requestType`: The type of data retrieval method from the server. See the request types sections for more information.
- `settings`: a set of key/value pairs that relate to the server communication technology. The following settings are available:
    - [ajax and ajax-poll] any settings found in jQuery.ajax()

- [ajax and ajax-poll] cache: see the cache section for more details.
- [ajax and ajax-poll] decoder: see the decoder section for more details
- [ajax-poll] frequency: the polling frequency in milliseconds.

```
amplify.request.define( string resourceId , function resource )
```

Define a custom request.

- `resourceId`: Identifier string for the resource.
- `resource`: Function to handle requests. Receives a hash with the following properties:
  - `resourceId`: Identifier string for the resource.
  - `data`: Data provided by the user.
  - `success`: Callback to invoke on success.
  - `error`: Callback to invoke on error.

Request Types

Built-in Types

`amplify.request` comes with built-in types of `ajax` and `ajax-poll`.

Custom Types

You can also create additional types by adding to the `amplify.request.types` hash. You can also define custom one-off types for single requests.

Data Handling

Pre-defined Data

When defining an ajax request, you can provide data in the definition. This data will be merged (via a deep extend) with any data provided with the request. Data provided with the request will override data provided in the definition.

You can define variables in the URL of an ajax request by wrapping the variable in curly braces, e.g., `"/user/{id}"`. The variable will be replaced by the respective value from the data provided in the request. Whenever a variable is replaced, the value is removed from the data (not submitted as GET/POST data). If there are variables that are not replaced, they will remain in the URL.

Decoders

Decoders allow you to parse an ajax response before calling the success or error callback. This allows you to return data marked with a status and react accordingly. This also allows you to manipulate the data any way you want before passing the data along to the callback.

Built-in Decoders

JSend is a built in decoder provided with the library.

Custom decoders

You can define new decoders by adding to the `amplify.request.decoders` hash. A popular use case for decoders is when you have a JSON envelope that must be unpacked with each response from the server.

You can also define custom one-off decoders for single requests, which is specified as a function in the settings hash for `amplify.request.define`.

```
amplify.request.decoders.decoderName =
    function( hash data, string status, object xhr,
        function success, function error )
```

Define a decoder. `decoderName` should be replaced with the decoder name of your choosing.

- `data`: Data returned from the ajax request.
- `status`: Status of the ajax request.
- `xhr`: xhr object used to make the request.
- `success`: Callback to invoke on success.
- `error`: Callback to invoke on error.

Cache

In-memory Cache

There is a built-in memory cache. You can pass a boolean to enable caching of a request, e.g., `cache: true`. You can also pass a number to specify that the response should be cached for a certain amount of time. For example, `cache: 30` will cache the response for 30 milliseconds.

- `cache: boolean` Cache the data in memory for the remainder of this page load.
- `cache: number` Cache the data in memory for the specified number of milliseconds.

Named Caches

- `cache: string` Cache the data using a pre-defined caching mechanism.

You can also persist a cache if `amplify.store` has been loaded. You can specify `cache: "persist"` to cache in the default store or you can specify any of the specific stores available, e.g., `cache: "localStorage"`.

Custom Cache

You can also create additional cache types by adding to the `amplify.request.cache` hash.

```
amplify.request.cache.customCacheName = function( hash resource, hash settings, hash ajaxSettings )
```

Definition for a caching mechanism. `customCacheName` should be replaced with the custom name of your choosing.

- `resource`: The definition of the resource being requested.
- `settings`: The settings for the request.
- `ajaxSettings`: The settings that will be passed to `jQuery.ajax()`.

Examples

The examples assume that the request location returns the following as json unless specified otherwise:

```
{
    "foo" : "bar",
    "baz" : "qux"
}
```

Set up and use a request utilizing Ajax

```
amplify.request.define( "ajaxExample1", "ajax", {
    url: "/myApiUrl",
    dataType: "json",
    type: "GET"
});

// later in code
amplify.request( "ajaxExample", function( data ) {
    data.foo; // bar
});
```

Set up and use a request utilizing Ajax and Caching

```
amplify.request.define( "ajaxExample2", "ajax", {
    url: "/myApiUrl",
    dataType: "json",
    type: "GET",
    cache: "persist"
});

// later in code
amplify.request( "ajaxExample2", function( data ) {
    data.foo; // bar
});

// a second call will result in pulling from the cache
amplify.request( "ajaxExample2", function( data ) {
    data.baz; // qux
})
```

Set up and use a RESTful request utilizing Ajax

```
amplify.request.define( "ajaxRESTFulExample", "ajax", {
    url: "/myRestFulApi/{type}/{id}",
    type: "GET"
})

// later in code
amplify.request( "ajaxRESTFulExample",
    {
        type: "foo",
        id: "bar"
    },
    functon( data ) {
        // /myRESTFulApi/foo/bar was the URL used
        data.foo; // bar
    }
);
```

POST data with Ajax

```
amplify.request.define( "ajaxPostExample", "ajax", {
    url: "/myRestFulApi",
    type: "POST"
})

// later in code
amplify.request( "ajaxPostExample",
    {
        type: "foo",
        id: "bar"
    },
    functon( data ) {
        data.foo; // bar
    }
);
```

Setting up and using decoders

This example assumes the following envelope format:

Success:

```
{
    "status": "success",
    "data" : {
        "foo": "bar",
        "baz": "qux"
    }
}
```

Fail (or Error):

```
{
    "status": "fail",
    "message": "failure message."
}
```

Example:

```
amplify.request.decoders.appEnvelope =
    function ( data, status, xhr, success, error ) {
        if ( data.status === "success" ) {
            success ( data.data );
        } else if ( data.status === "fail" || data.status === "error" ) {
            error( data.message, data.status );
        } else {
            error( data.message , "fatal" );
        }
    };

amplify.request.define( "decoderExample", "ajax", {
    url: "/myAjaxUrl",
    type: "POST",
    decoder: "appEnvelope"
});

amplify.request({
    resourceId: "decoderExample",
    success: function( data ) {
        data.foo; // bar
    },
    error: function( message, level ) {
        alert( "always handle errors with alerts." );
    }
});
```

POST with caching and single-use decoder

This example assumes the following envelope format:

Success:

```
{
    "status": "success",
    "data" : {
        "foo": "bar",
        "baz": "qux"
    }
}
```

Fail (or Error):

```
{
    "status": "fail",
    "message": "failure message."
}
```

Example:

```
amplify.request.define( "decoderSingleExample", "ajax", {
    url: "/myAjaxUrl",
    type: "POST",
    decoder: function ( data, status, xhr, success, error ) {
        if ( data.status === "success" ) {
            success ( data.data );
        } else if ( data.status === "fail" || data.status === "error" ) {
            error( data.message, data.status );
        } else {
            error( data.message , "fatal" );
        }
    }
});

amplify.request({
    resourceId: "decoderSingleExample",
    success: function( data ) {
        data.foo; // bar
    },
    error: function( message, level ) {
        alert( "always handle errors with alerts." );
    }
});
```