API Builder Connector Getting Started Guide

Introduction

This guide covers the basics to create and build a Connector. A Connector is a component of a project that allows you to access data from an external source. A Connector is set up the same as a Builder project except you have logic for your connector in the `lib` folder and do not have API, Block or Web components.

For this example, the connector will integrate with the Twitter REST API. You will need to create a new Twitter app and generate the OAuth access tokens. To create a new Twitter app, go to https://apps.twitter.com/, then generate the access token from the app's Keys and Access Tokens tab.

Create a connector

To create a new connector, from your workspace directory, execute the `appc generate` command. When prompted, select **Component** for the type of component, **Connector** for the component to generate, and **Empty Connector Project** for the connector to generate. Enter `com.connector.twitter` as the name and directory name for your project.

```
$ appc generate
Appcelerator Command-Line Interface, version 0.2.230
Copyright (c) 2014-2015, Appcelerator, Inc. All Rights Reserved.

? What type of component would you like to generate? Arrow Component
? What Arrow component would you like to generate? Arrow Connector
? Which Connector would you like to generate? Empty Connector Project
? What is the connector name? com.connector.twitter
? Which directory to generate into? com.connector.twitter
```

Add dependencies

As with all NPM modules, add your dependencies to the `dependencies` object in the `package.json` file. This example connector will be utilizing the twitter NPM module to make calls to Twitter. Add the `twitter` key with a `~1.2.5` value to the `dependencies` object.

package.json

```
{ ...
  "dependencies" : {
    ...
    "twitter": "~1.2.5"
  }
 ...
}
```

Configuration file

The boilerplate sample contains the `conf/default.js` configuration file. This file is used for testing the connector. You will also need to create a configuration file called `example.config.js`. When you install the connector, this file is copied to the project's `conf` directory. You need to add settings to configure your connector in these files, specifically add the settings to the `connectors` object. For the Twitter connector, it will need the app keys, OAuth tokens, and an account name of the Twitter feed to access. Create the `conf/example.config.js` file, then add the content below to it. You will also need to add the following keys to the connectors object in the `conf/default.js` file and add your Twitter app information to it for testing purposes.

./conf/example.config.js

```
module.exports = {
  connectors: {
    'com.connector.twitter': {
      account: 'TWITTER_ACCOUNT',
      consumer_key: 'TWITTER_API_KEY',
      consumer_secret: 'TWITTER_API_SECRET',
      access_token_key: 'TWITTER_ACCESS_TOKEN',
      access_token_secret: 'TWITTER_ACCESS_TOKEN_SECRET'
    }
  }
}
```

Initializer file

The `app.js` file contains code that initializes the connector when it is used as a server for testing. You can hook into the lifecycle events of the server. The boilerplate file contains logic that is used to create a model. Update the code to let the application create a connector and access the tweet's text response parameter as the `status` parameter. Later on, you will add logic to automatically create a model when a connector is installed on an application.

app.js

```
var Arrow = require('arrow'),
    server = new Arrow();

server.addModel(Arrow.Model.extend('tweet', {
  fields: {
    status: { type: String, name: 'text' }
  },
  connector: 'com.connector.twitter'
}));

server.start();
```

Develop the connector

API Builder will guide you through the rest of the process. To start developing your connector, run the project in one console window, then edit the files with the connector logic in another console or editor. As you save your files, API Builder will automatically update your connector and restart the server instance, allowing you to work on and test the connector incrementally.

Let's get started. First, run the project and look at the log output:

```
appc run
...
This connector does not do much of anything at the moment.
Why don't you take a look at the "capabilities" array in:
/Users/jdoe/workspace/com.connector.twitternew/lib/index.js
```

Follow the log instructions and open the ./lib/index.js file. You will see an exposed create function that uses the `Connector.extend()` method to create a new connector. In the object passed to the `extend()` method, you will see a capabilities field with an array of `Capability` constants. The constants are used to indicate to API Builder, which operations the connector can perform. We will add capabilities incrementally.

Let's start with connecting to a datasource.

1. Add the `defaultConfig` field to the parameter object passed to the `extend()` method. The `defaultConfig` field tells API Builder to copy the example configuration file to the project's `conf` directory when the user adds the connector to the project.
2. Uncomment the line containing `Capabilities.ConnectsToADataSource`.
3. Add the `postCreate()` function to attach a helper function to the connector instance that can be used by other connector methods.

./lib/index.js Expand source

```
...
exports.create = function (Arrow) {
  ...
  return Connector.extend({
    defaultConfig: require('fs').readFileSync(__dirname +
'/../conf/example.config.js', 'utf8'),
    ...
    capabilities: [
      Capabilities.ConnectsToADataSource,

      // TODO: Each of these capabilities is optional; add the ones you want, and
delete the rest.
      // (Hint: I've found it to be easiest to add these one at a time, running
`appc run` for guidance.)
      //Capabilities.ValidatesConfiguration,
      //Capabilities.ContainsModels,
      //Capabilities.GeneratesModels,
      //Capabilities.CanCreate,
      //Capabilities.CanRetrieve,
      //Capabilities.CanUpdate,
      //Capabilities.CanDelete,
      //Capabilities.AuthenticatesThroughConnector
    ],
    postCreate: function() {
      this.createInstance = function (Model, data) {
        var model = Model.instance(data, true);
        model.setPrimaryKey(data.id_str);
        return model;
      };
    }
  });
};
```

4. Save the file. You will see the following log output:

```
The "ConnectsToADataSource" capability has been enabled, so we need to make a
couple of changes:
 - Created `lib/lifecycle/connect.js` (contains 2 TODOs)
 - Created `lib/lifecycle/disconnect.js` (contains 2 TODOs)

Please go take a look at the TODOs in these new files, then do an `appc run` or
`npm test` to try out the new capabilities.
```

API Builder generates some new files in the `lib` folder. You will need to implement some logic to connect to and disconnect from the datasource.

Add connect logic

Open `./lib/lifecycle/connect.js` and replace the contents of the file with the following. The `connect.js` file contains logic to implement the connector's `connect()` method, which authorizes the connector to talk to the Twitter APIs.

Next, let's implement some logic to retrieve the tweets.

Add retrieve logic

To allow the connector to retrieve model data, implement the `CanRetrieve` capability.

1. Open the `./lib/index.js` file, uncomment the line containing `Capabilities.CanRetrieve`, and save the file. The log output tells us to update four new files that API Builder just generated.

```
The "CanRetrieve" capability has been enabled, so we need to make a couple of
changes:
 - Created `lib/methods/distinct.js` (contains 3 TODOs)
 - Created `lib/methods/findAll.js` (contains 3 TODOs)
 - Created `lib/methods/findById.js` (contains 4 TODOs)
 - Created `lib/methods/query.js` (contains 5 TODOs)
(Hint: If you only want to support some of these methods, feel free to delete the
others.)
Please go take a look at the TODOs in these new files, then do an `appc run` or
`npm test` to try out the new capabilities.
```

2. Remove the `distinct.js` and `query.js` files. We will not be implementing these for the example.

./lib/lifecycle/connect.js

```
var Twitter = require('twitter');

exports.connect = function (next) {
  // Initialize the client
  // Use this.config to get values from the configuration file
  this.client = new Twitter({
    consumer_key: this.config.consumer_key,
    consumer_secret: this.config.consumer_secret,
    access_token_key: this.config.access_token_key,
    access_token_secret: this.config.access_token_secret
  });
  next();
};
```

3. Open `./lib/methods/findAll.js` and replace the contents of the file with the following. The `findAll.js` file implements the connector's `findAll()` method, which retrieves all tweets from the user's Twitter account, or at least, as many tweets as Twitter will allow us to fetch.

./lib/methods/findAll.js

```
var Arrow = require('arrow'),
  Collection = Arrow.Collection,
  ORMError = Arrow.ORMError;

exports.findAll = function findAll(Model, callback) {
    var params = {screen_name: this.config.account};
    // this Twitter API only returns the last twenty tweets
    this.client.get('statuses/user_timeline', params, function(error, tweets,
response) {
    if (!error) {
      var results = [];
      for (var i = 0; i < tweets.length; i++) {
        results.push(Model.connector.createInstance(Model, tweets[i]));
      }
      callback(null, new Collection(Model, results));
    } else {
      this.logger.error(error);
      callback(new ORMError('ERROR: Could not fetch tweets!'));
    }
  });
};
```

4. Open `./lib/methods/findById.js` and replace the contents of the file with the following. The `findById.js` file implements the connector's `findById()` method, which retrieves one specific tweet, identified by its ID, from the datasource.

./lib/methods/findById.js

```
var Arrow = require('arrow'),
  ORMError = Arrow.ORMError;

exports.findById = function (Model, id, callback) {
  this.client.get('statuses/show/' + id, {}, function(error, tweet, response) {
    if (!error) {
      callback(null, Model.connector.createInstance(Model, tweet));
    } else {
      this.logger.error(error);
      callback(new ORMError('ERROR: Could not fetch tweet!'));
    }
  });
};
```

Let's see if the connector's retrieve logic works.

In a web browser:

1. Navigate to the Admin Console. You can access it using the following URL: `http://localhost:8080/api-create-ui`
2. In the top navigation bar, click the **Data** tab.
3. In the left navigation bar, click **tweet**.

The Admin console will retrieve and display the user's tweets.

Next, let's add some logic to create a tweet.

Add create logic

To allow the connector to create data, implement the `CanCreate` capability.

1. Open the `./lib/index.js` file, uncomment the line containing `Capabilities.CanCreate` and save the file. You will see the following console output:

```
The "CanCreate" capability has been enabled, so we need to make a couple of
changes:
 - Created `lib/methods/create.js` (contains 3 TODOs)
Please go take a look at the TODOs in these new files, then do an `appc run` or
`npm test` to try out the new capabilities.
```

2. Open `./lib/methods/create.js` and replace the contents of the file with the following. The `create.js` file implements the connector's `create()` method, which allows the connector to post a tweet on the user's Twitter feed.

./lib/methods/create.js

```
var Arrow = require('arrow'),
  ORMError = Arrow.ORMError;

exports.create = function (Model, values, callback) {
  var params = {'status': values.text};
  this.client.post('statuses/update', params, function(error, tweet, response) {
    if (!error) {
      callback(null, Model.connector.createInstance(Model, tweet));
    } else {
      callback(new ORMError('ERROR: Unable to create tweet!'));
    }
  });
};
```

Let's test the create logic.

In the Admin console, you should still see the list of tweets. If not, click **Data** in the top navigation bar, then click **tweet** in the left navigation bar.

1. Click the **Add (+)** button in the top-right corner of the console. A dialog will appear.
2. Enter a status message, then click **OK**.

The new status message will be displayed in the list of tweets.

Next, let's add some logic to remove the tweets.

Add delete logic

To allow the connector to remove data, implement the `CanDelete` capability.

1. Open the `./lib/index.js` file, uncomment the line containing `Capabilities.CanDelete` and save the file. You will see the following console output:

```
The "CanDelete" capability has been enabled, so we need to make a couple of
changes:
 - Created `lib/methods/delete.js` (contains 4 TODOs)
 - Created `lib/methods/deleteAll.js` (contains 3 TODOs)
(Hint: If you only want to support some of these methods, feel free to delete the
others.)
Please go take a look at the TODOs in these new files, then do an `appc run` or
`npm test` to try out the new capabilities.
```

2. Remove the `lib/methods/deleteAll.js`.
3. Open `./lib/methods/delete.js` and replace the contents of the file with the following. The `delete.js` file implements the connector's `delete()` method, which allows the connector to delete a specific tweet, specified by its ID, from the user's Twitter feed.

./lib/methods/delete.js

```
var Arrow = require('arrow'),
  ORMError = Arrow.ORMError;

exports['delete'] = function (Model, instance, callback) {
  this.client.post('statuses/destroy/' + instance.id + '.json', {},
function(error, tweet, response) {
    if (!error) {
      callback(null, Model.connector.createInstance(Model, tweet));
    } else {
      callback(new ORMError('ERROR: Could not delete tweet!'));
    }
  });
};
```

Let's test the delete logic.

In the Admin console, you should still see the list of tweets. If not, click **Data** in the top navigation bar, then click **tweet** in the left navigation bar.

1. Click the tweet you previously added to the list. A dialog appears.
2. Click **Delete**.

The new status message will be removed from the list of tweets.

Setup a model

To have a model included with your connector, implement the `ContainModel` capability.

1. Remove the `server.addModel` method from the `app.js` file.
2. Open the `./lib/index.js` file, uncomment the line containing `Capabilities.ContainModel` and save the file. You will see the following console output:

```
The "ContainsModels" capability has been enabled, so we need to make a couple of
changes:
 - Created `models/yourModel.js` (contains 2 TODOs)
```

3. Rename the `yourModel.js` file to `tweet.js`.
4. Open `./models/tweet.js` and replace the contents of the file with the following:

./models/tweet.js

```
var Arrow = require('arrow');

var Tweet = Arrow.Model.extend('tweet', {
  fields: {
    status: { type: String, name: 'text' }
  },
  connector: 'com.connector.twitter'
});

module.exports = Tweet;
```

5. Reload the Admin console and retest adding and removing a model.

Publish the connector

To publish the connector, execute the following command from the project directory:

```
appc publish
```

By default, the access level for the connector is set to private, so only the creator can access the connector. To share the connector with other people or publicly, specify a different access level with the `appc access` command and add people or organizations to your component using the `appc user` and `appc org` commands.

Next steps

For information about installing your connector, see Add a Connector.

For information about creating a connector, see API Builder Connector Project.