dbDriver

This configuration property specifies the default database driver to use for Profound.js Record Level Access and SQL operations.

Click here for information on how to modify the 'dbDriver' setting.

Profound.js database drivers

The following is a list of database drivers shipped with Profound.js.

Profound.js Database Driver	dbDriver Name	Record Level Access	SQL	Required Components	Supported API
IBM i DB2	"IBMi"	Yes	Yes	Profound.js Connector on IBM i	All Record Level Access API, All SQL API
Offline JSON Store (see offline capabilities)	"jsonDB"	Yes	No	No extra components required	Record Level Access API (No commitment control or record locking)
MySQL / MariaDB	"mysql"	Yes	Yes	Profound.js Connector for MySQL and mysql npm package	All SQL API Record Level Access API, (No commitment control, record locking, or access by RRN)
Microsoft SQL Server	"mssql"	Yes	Yes	Profound.js Connector for MS SQL Server and mssql npm package	All SQL API Record Level Access API, (No commitment control, record locking, or access by RRN)
Oracle	"oracled b"	Yes	Yes	Profound.js Connector for Oracle and oracledb npm package	All SQL API Record Level Access API, (No commitment control, record locking, or access by RRN)

For IBM i DB2, if the IBM i system is remote in relation to the Profound.js server and if you're not starting your session from Genie, then you should also set the connectorURL and connectorCredentials configuration settings.

For MySQL / MariaDB, Microsoft SQL Server, and Oracle, you should set the connectionDetails configuration setting. Connection details can also be specified using the pjs.connect() API.

Other Profound.js drivers can be defined by extending the db.drivers object attached to the profoundjs package module exports.

The Profound.js driver defined by this configuration property is used by these API:

- Record Level Access API
- SQL API

Using multiple database drivers in one application

There is only one default dbDriver for any one installation of Profound.js; however, it is possible to use alternate database connections to other databases from one Profound.js application.

For Record Level Access, an alternate driver can be specified using the pjs.defineTable() "driver" configuration parameter.

For SQL, an alternate driver cannot be specified to be used by the Profound.js SQL API; however, you can use the corresponding Node.js package directly from Profound.js modules. For example, if you wanted to access a MySQL database from an instance configured with the IBM i DB2 driver, you can find a compatible MySQL database driver package on npm, such as this one: https://www.npmjs.com/package/mysql.

Be aware that most npm packages will provide examples using asynchronous calls, whereas Profound.js uses a top-down paradigm which makes business programming a lot easier. This is accomplished through a built-in component called Fibers. To make asynchronous calls procedural, you must use one of the Fibers API provided. Most tasks can be handled with the pjs.fiber.wrap() API.

So, for example, the MySQL package on npm provides this sample statement:

```
connection.query('SELECT 1 + 1 AS solution', function (error, results) {
  if (error) throw error;
  console.log('The solution is: ', results[0].solution);
});
```

The results are returned into a separate function, which runs asynchronously. There is no guarantee that any code placed after the call to connection.query() will run after the database request completes.

In Profound.js, the equivalent code might look as follows:

```
var query = pjs.fiber.wrap(connection.query, connection);
var results = query(connection.query, 'SELECT 1 + 1 AS solution');
console.log('The solution is: ', results[0].solution);
```

The code runs in a more predictable top-down manner. A separate asynchronous function is not required.

Connecting to other databases

Connecting to databases for which a driver does not exist is accomplished through standard NPM packages and Fibers.

For example, a Profound.js module to connect to a MongoDB database and retrieve certain documents from this database might look as follows:

```
// Bring in Mongo Client from the 'mongodb' NPM package
const MongoClient = require('mongodb').MongoClient;

// Connection URL
const url = 'mongodb://localhost:27017';

// Database Name
const dbName = 'myproject';

function getMongoDocs() {
  var connect = pjs.fiber.wrap(MongoClient.connect, MongoClient);
  var client = connect(url);
  var db = client.db(dbName);
  var collection = db.collection('documents');
  var findDocs = collection.find({'someproperty': 'somevalue'});
  var findDocsToArray = pjs.fiber.wrap(findDocs.toArray, findDocs);
  var docs = findDocsToArray();

  console.log(docs);

  client.close();
  return docs;
}

exports.run = getMongoDocs;
```