Calling System API and Service Program Procedures

Profound.js provides a number of simple ways to call any IBM i system API or any custom ILE service program procedure. This page describes the different ways this can be done.

By defining a callable function

The easiest way to define the procedure as a callable function in JavaScript is with the use of the pjs.defineProc() API. It is simply a matter of specifying all parameters and a return value if the procedure returns one. For example:

```
// Define external procedure
pjs.defineProc("sqrt", {
  srvpgm: "QSYS/QC2UTIL1",
  parms: [
    { type: 'float', length: 8 }
  ],
  result: { type: 'float', length: 8 }
});


// Use external procedure as a simple function
console.log(sqrt(16));  // output is 4 (the square root of 16)
```

If the service program is converted to Node.js in the future, no dependencies will be broken, and you can continue to call the functions in the same exact way.

By calling procedures directly or creating wrapper functions

With the pjs.callProc() API, you can call IBM i procedures directly. It also makes it easy to create a JavaScript function wrapper for any ILE subprocedure. Once the wrapper is created, the system API or subprocedure can again be called like a simple JavaScript function.

Creating the wrapper is really easy thanks to strong data type support in Profound.js. It is simply a matter of defining the parameters and return values as fields with the appropriate IBM i data type.

A sample wrapper function looks like this:

```
function myproc(parm1, parm2) {
  pjs.define("parm1", { type: 'char', length: 10, parm: parm1 });
  pjs.define("parm2", { type: 'char', length: 20, refParm: parm2 });

  pjs.callProc({
    srvpgm: "MYLIB/MYSRVPGM",
    procedure: "MYPROC",
    arguments: [
      { field: "parm1" },
      { field: "parm2", byRef: true }
    ]
  });
}
```

You can then call the function using something like this:

```
myproc("constant", pjs.refParm("field"));
```

In this example, the first parameter is passed by value and the second parameter is passed by reference.

If the procedure itself is converted to Node.js in the future, no dependencies will be broken, and you can continue to call the function in the same exact way.

By using pjs.callProcedure() for advanced scenarios

For other more advanced use cases, Profound.js also provides the pjs.callProcedure() API. With pjs.callProcedure(), you can work with parameters and return values at the buffer level and provide details such as operational parameter descriptors.