MeshAgent / modules / win-dispatcher.js

359 lines (326 loc) · 12.8 KB

Code Blame Raw

```
1   /*
2   Copyright 2019-2022 Intel Corporation
3
4   Licensed under the Apache License, Version 2.0 (the "License");
5   you may not use this file except in compliance with the License.
6   You may obtain a copy of the License at
7
8       http://www.apache.org/licenses/LICENSE-2.0
9
10  Unless required by applicable law or agreed to in writing, software
11  distributed under the License is distributed on an "AS IS" BASIS,
12  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  See the License for the specific language governing permissions and
14  limitations under the License.
15  */
16
17
18  //
19  // win-dispatcher is used as a helper function to be able to dispatch
20  // code to be executed by a child process, by way of using an IPC to interact
21  // with the child process
22  //
23
24  //
25  // This was an anonymous function that was pulled out, so that the
26  // JS runtime would not try to create strong references to parent scoped objects,
```

```
27      // when the anonymous function was used as a function callback
28      //
29      function empty_func()
30      {
31          var p = this.parent;
32          if (p != null)
33          {
34              if (p._ipc) { p._ipc.parent = null };
35              if (p._ipc2) { p._ipc2.parent = null; }
36              if (p._client) { p._client._parent = null; }
37              p._client = null;
38              if (p._control) { p._control._parent = null; }
39              p._control = null;
40              p = null;
41          }
42      }
43
44      //
45      // This was an anonymous function that was pulled out, so that the
46      // JS runtime would not try to create strong references to parent scoped objects,
47      // when the anonymous function was used as a function callback
48      //
49      function empty_func2()
50      {
51      }
52
53      //
54      // This function sends a command via IPC to the child process to invoke an action
55      //
56      function ipc_invoke(method, args)
57      {
58          var d, h = Buffer.alloc(4);
59          d = Buffer.from(JSON.stringify({ command: 'invoke', value: { method: method, args: args } }));
60          h.writeUInt32LE(d.length + 4);
61          this._control.write(h);
62          this._control.write(d);
63      }
64
65      function ipc1_finalized()
66      {
67          //console.log('IPC1 Finalized');
68      }
69      function ipc2_finalized()
70      {
71          //console.log('IPC2 Finalized');
72      }
```

```
73      function ipc1_server_finalized()
74      {
75          //console.log('IPC1 Server Finalized');
76      }
77      function ipc2_server_finalized()
78      {
79          //console.log('IPC2 Server Finalized');
80      }
81
82      //
83      // Secondary Connection handler function that is called on IPC connection, to initialize some back
84      //
85      function ipc2_connection(s)
86      {
87          this.parent._control = s;
88          this.parent._control._parent = this;
89          this.close();
90          this.parent.invoke = ipc_invoke;
91          s.on('end', empty_func2); // DO NOT DELETE this line!
92          s.on('~', ipc2_finalized);
93      }
94
95      //
96      // Primary Connection handler function that is called on IPC connection, that is used to initialize
97      //
98      function ipc_connection(s)
99      {
100         this.parent._client = s;
101         this.parent._client._parent = this;
102         this.close();
103         var d, h = Buffer.alloc(4);
104         s.descriptorMetadata = 'win-dispatcher, ' + this.parent.options.launch.module + '.' + this.pare
105
106         for (var m in this.parent.options.modules)
107         {
108             // Enumerate each module passed in, and pass it along to the child via IPC
109             d = Buffer.from(JSON.stringify({ command: 'addModule', value: { name: this.parent.options.m
110             h.writeUInt32LE(d.length + 4);
111             s.write(h);
112             s.write(d);
113         }
114
115         // Launch the specified module/function via IPC
116         d = Buffer.from(JSON.stringify({ command: 'launch', value: { split: this.parent.options.launch.
117         h.writeUInt32LE(d.length + 4);
118         s.write(h);
```

```
                s.write(n);
```

```
285     {
286         var ipcPath = '\\\\.\\pipe\\taskRedirection-' + ipc;
287         global.ipc2Client = require('net').createConnection({ path: ipcPath + 'C' }, function ()
288         {
289             //
290             // This is the secondary channel, that is used as a control channel after the child operati
291             //
292             this.on('data', function (c)
293             {
294                 var cLen = c.readUInt32LE(0);
295                 if (cLen > c.length)
296                 {
297                     this.unshift(c);
298                     return;
299                 }
300                 var cmd = JSON.parse(c.slice(4, cLen).toString());
301                 switch (cmd.command)
```

```
302                 {
303                     case 'invoke':
304                         global._proxyStream[cmd.value.method].apply(global._proxyStream, cmd.value.args
305                         break;
306                 }
307 
308                 if (cLen < c.length) { this.unshift(c.slice(cLen)); }
309             });
310         });
311         global.ipcClient = require('net').createConnection({ path: ipcPath }, function ()
312         {
313             //
314             // This is the primary IPC channel. It is used to establish/initialize what will run in the
315             // It will ultimately result in a stream object being piped to whatever function is launche
316             //
317             this.on('close', function () { process.exit(); });
318             this.on('data', function (c)
319             {
320                 var cLen = c.readUInt32LE(0);
321                 if (cLen > c.length)
322                 {
323                     this.unshift(c);
324                     return;
325                 }
326                 var cmd = JSON.parse(c.slice(4, cLen).toString());
327                 switch (cmd.command)
328                 {
329                     case 'addModule':
330                         addModule(cmd.value.name, cmd.value.js);        // Adds a JS module to the modu
331                         break;
332                     case 'launch':                                      // Launches the specified modul
333                         var obj = require(cmd.value.module);
334                         global._proxyStream = obj[cmd.value.method].apply(obj, cmd.value.args);
335                         if (cmd.value.split)
336                         {
337                             global._proxyStream.out.pipe(this, { end: false });
338                             this.pipe(global._proxyStream.in, { end: false });
339                             global._proxyStream.out.on('end', function () { process.exit(); });
340                         }
341                         else
342                         {
343                             global._proxyStream.pipe(this, { end: false });
344                             this.pipe(global._proxyStream, { end: false });
345                             global._proxyStream.on('end', function () { process.exit(); });
346                         }
347                         this.on('end', function () { process.exit(); });
```

```
348                         break;
349                 }
350
351                 if (cLen < c.length) { this.unshift(c.slice(cLen)); }
352             });
353         });
354         global.ipcClient.on('error', function () { process.exit(); });
355         global.ipc2Client.on('error', function () { process.exit(); });
356     }
357
358     module.exports = { dispatch: dispatch, connect: connect };
```