MeshAgent / modules / win-dispatcher.js

krayon007 Added documentation 705aac1 · 2 years ago History

```
/*
Copyright 2019-2022 Intel Corporation

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/


//
// win-dispatcher is used as a helper function to be able to dispatch
// code to be executed by a child process, by way of using an IPC to interact
// with the child process
//

//
// This was an anonymous function that was pulled out, so that the
// JS runtime would not try to create strong references to parent scoped objects,
// when the anonymous function was used as a function callback
//
function empty_func()
{
    var p = this.parent;
    if (p != null)
    {
        if (p._ipc) { p._ipc.parent = null };
        if (p._ipc2) { p._ipc2.parent = null; }
        if (p._client) { p._client._parent = null; }
        p._client = null;
        if (p._control) { p._control._parent = null; }
        p._control = null;
        p = null;
    }
}

//
// This was an anonymous function that was pulled out, so that the
// JS runtime would not try to create strong references to parent scoped objects,
// when the anonymous function was used as a function callback
//
function empty_func2()
{
}

//
// This function sends a command via IPC to the child process to invoke an action
//
function ipc_invoke(method, args)
```

```
{
    var d, h = Buffer.alloc(4);
    d = Buffer.from(JSON.stringify({ command: 'invoke', value: { method: method, args:
    h.writeUInt32LE(d.length + 4);
    this._control.write(h);
    this._control.write(d);
}

function ipc1_finalized()
{
    //console.log('IPC1 Finalized');
}
function ipc2_finalized()
{
    //console.log('IPC2 Finalized');
}
function ipc1_server_finalized()
{
    //console.log('IPC1 Server Finalized');
}
function ipc2_server_finalized()
{
    //console.log('IPC2 Server Finalized');
}

//
// Secondary Connection handler function that is called on IPC connection, to initializ
//
function ipc2_connection(s)
{
    this.parent._control = s;
    this.parent._control._parent = this;
    this.close();
    this.parent.invoke = ipc_invoke;
    s.on('end', empty_func2); // DO NOT DELETE this line!
    s.on('~', ipc2_finalized);
}

//
// Primary Connection handler function that is called on IPC connection, that is used t
//
function ipc_connection(s)
{
    this.parent._client = s;
    this.parent._client._parent = this;
    this.close();
    var d, h = Buffer.alloc(4);
    s.descriptorMetadata = 'win-dispatcher, ' + this.parent.options.launch.module + '.'

    for (var m in this.parent.options.modules)
    {
        // Enumerate each module passed in, and pass it along to the child via IPC
        d = Buffer.from(JSON.stringify({ command: 'addModule', value: { name: this.pare
        h.writeUInt32LE(d.length + 4);
        s.write(h);
        s.write(d);
    }

    // Launch the specified module/function via IPC
    d = Buffer.from(JSON.stringify({ command: 'launch', value: { split: this.parent.opt
    h.writeUInt32LE(d.length + 4);
    s.write(h);
    s.write(d);
    s.on('~', ipc1_finalized);
    this.parent.emit('connection', s);
}

// Shutdown the IPC to the child. The child will detect this and shutdown as well.
function dispatcher_shutdown()
{
    this._ipc.close();
    this._ipc2.close();
    this._ipc = null;
    this._ipc2 = null;
}
```

```

//
// Dispatch an operation to a child process
//
function dispatch(options)
{
    // These are the minimum options that MUST be passed in
    if (!options || !options.modules || !options.launch || !options.launch.module || !o

    var ipcInteger
    var ret = { options: options };
    require('events').EventEmitter.call(ret, true).createEvent('connection');

    //
    // Create TWO IPC channels to the child process... The primary is used to implement
    // The secondary IPC channel is used as a "control channel" with the child process
    //
    ret._ipc = require('net').createServer(); ret._ipc.parent = ret;
    ret._ipc2 = require('net').createServer(); ret._ipc2.parent = ret;
    ret._ipc.on('close', empty_func);
    ret._ipc2.on('close', empty_func);
    ret._ipc.once('~', ipc1_server_finalized);
    ret._ipc2.once('~', ipc2_server_finalized);
```

MeshAgent / modules / win-dispatcher.js    Top

Files

52cf129

Go to file

- .github
- docs
- lib-jpeg-turbo
- meshconsole
- meshcore
- meshreset
- meshservice
- microscript
- microstack
- modules
  - utils
  - AgentHashTool.js
  - CSP.js
  - DeviceManager.js
  - MSH_Installer.js
  - PE_Parser.js
  - PostBuild.js
  - RecoveryCore.js
  - _agentNodeId.js
  - _agentStatus.js
  - agent-installer.js
  - agent-selftest.js
  - amt-lme.js
  - amt-mei.js
  - amt-scanner.js
  - amt-script.js
  - amt-wsman-duk.js
  - amt-wsman.js

Code    Blame    359 lines (326 loc) · 12.8 KB    Raw

```
        ret._ipcPath = '\\\\.\\pipe\\taskRedirection-' + ipcInteger;

        try
        {
            ret._ipc.listen({ path: ret._ipcPath, writableAll: true });
            ret._ipc2.listen({ path: ret._ipcPath + 'C', writableAll: true });
            break;
        }
        catch (x)
        {
        }
    }

    //
    // The child process will hide the console, and then initalize as a client to the p
    //
    var str = Buffer.from("require('win-console').hide();require('win-dispatcher').conn
    ret._ipc2.once('connection', ipc2_connection);
    ret._ipc.once('connection', ipc_connection);
    ret.close = dispatcher_shutdown;

    try
    {
        //
        // Try to fetch user/domain settings to configure the child process
        //
        var user = null;
        var domain = null;
        if(options.user == null)
        {
            //
            // If no user was specified, we'll use the same user as the parent
            //
            if (require('user-sessions').getProcessOwnerName(process.pid).tsid == 0)
            {
                user = 'SYSTEM'
            }
            else
            {
                var info = require('user-sessions').getProcessOwnerName(process.pid);
                user = info.name;
                domain = info.domain;
            }
        }
        else
        {
            var u = options.user;
            if (u[0] == '"') { u = u.substring(1, u.length - 1); }
```

```
                var tokens = u.split('\\');
                if(tokens.length!=2) { throw('invalid user format');}
                user = tokens[1];
                domain = tokens[0];
            }

            console.info1('user- ' + user, 'domain- ' + domain);

            //
            // Use the windows scheduler to schedule the child process to run as the specif
            //
            var task = { name: 'MeshUserTask', user: user, domain: domain, execPath: proces
            require('win-tasks').addTask(task);
            require('win-tasks').getTask({ name: 'MeshUserTask' }).run();
            require('win-tasks').deleteTask('MeshUserTask');
            return (ret);
        }
        catch(xx)
        {
            console.info1(xx);
        }

        //
        // If we get here, it means we were unable to use the Windows Task Schedular COM AP
        // fallback to using SCHTASKS instead
        //
        console.info1('Using SCHTASKS...');

        var taskoptions = { env: { _target: process.execPath, _args: '-b64exec ' + str, _us
        for (var c1e in process.env)
        {
            taskoptions.env[c1e] = process.env[c1e];
        }

        //
        // We're going to use Windows Powershell to schedule the task, because there are a
        // also specify which cannot be set directly with SCHTASKS
        //
        var child = require('child_process').execFile(process.env['windir'] + '\\System32\\
        child.stderr.on('data', empty_func2);
        child.stdout.on('data', empty_func2);
        child.stdin.write('SCHTASKS /CREATE /F /TN MeshUserTask /SC ONCE /ST 00:00 ');
        if (options.user)
        {
            child.stdin.write('/RU $env:_user ');
        }
        else
        {
            if (require('user-sessions').getProcessOwnerName(process.pid).tsid == 0)
            {
                // LocalSystem
                child.stdin.write('/RU SYSTEM ');
            }
        }
        child.stdin.write('/TR "$env:_target $env:_args"\r\n');
        child.stdin.write('$ts = New-Object -ComObject Schedule.service\r\n');
        child.stdin.write('$ts.connect()\r\n');
        child.stdin.write('$tsfolder = $ts.getfolder("\\")\r\n');
        child.stdin.write('$task = $tsfolder.GetTask("MeshUserTask")\r\n');
        child.stdin.write('$taskdef = $task.Definition\r\n');
        child.stdin.write('$taskdef.Settings.StopIfGoingOnBatteries = $false\r\n');
        child.stdin.write('$taskdef.Settings.DisallowStartIfOnBatteries = $false\r\n');
        child.stdin.write('$taskdef.Actions.Item(1).Path = $env:_target\r\n');
        child.stdin.write('$taskdef.Actions.Item(1).Arguments = $env:_args\r\n');
        child.stdin.write('$tsfolder.RegisterTaskDefinition($task.Name, $taskdef, 4, $null,

        child.stdin.write('SCHTASKS /RUN /TN MeshUserTask\r\n');
        child.stdin.write('SCHTASKS /DELETE /F /TN MeshUserTask\r\nexit\r\n');

        child.waitExit();
        return (ret);
    }

    //
```

```
// This function is called by the child process, so that it can act as client to the pa
// It contains all the logic to establish the two IPC channels
//
function connect(ipc)
{
    var ipcPath = '\\\\.\\pipe\\taskRedirection-' + ipc;
    global.ipc2Client = require('net').createConnection({ path: ipcPath + 'C' }, functi
    {
        //
        // This is the secondary channel, that is used as a control channel after the c
        //
        this.on('data', function (c)
        {
            var cLen = c.readUInt32LE(0);
            if (cLen > c.length)
            {
                this.unshift(c);
                return;
            }
            var cmd = JSON.parse(c.slice(4, cLen).toString());
            switch (cmd.command)
            {
                case 'invoke':
                    global._proxyStream[cmd.value.method].apply(global._proxyStream, cm
                    break;
            }

            if (cLen < c.length) { this.unshift(c.slice(cLen)); }
        });
    });
    global.ipcClient = require('net').createConnection({ path: ipcPath }, function ()
    {
        //
        // This is the primary IPC channel. It is used to establish/initialize what wil
        // It will ultimately result in a stream object being piped to whatever functio
        //
        this.on('close', function () { process.exit(); });
        this.on('data', function (c)
        {
            var cLen = c.readUInt32LE(0);
            if (cLen > c.length)
            {
                this.unshift(c);
                return;
            }
            var cmd = JSON.parse(c.slice(4, cLen).toString());
            switch (cmd.command)
            {
                case 'addModule':
                    addModule(cmd.value.name, cmd.value.js);        // Adds a JS module
                    break;
                case 'launch':                                      // Launches the spe
                    var obj = require(cmd.value.module);
                    global._proxyStream = obj[cmd.value.method].apply(obj, cmd.value.ar
                    if (cmd.value.split)
                    {
                        global._proxyStream.out.pipe(this, { end: false });
                        this.pipe(global._proxyStream.in, { end: false });
                        global._proxyStream.out.on('end', function () { process.exit();
                    }
                    else
                    {
                        global._proxyStream.pipe(this, { end: false });
                        this.pipe(global._proxyStream, { end: false });
                        global._proxyStream.on('end', function () { process.exit(); });
                    }
                    this.on('end', function () { process.exit(); });
                    break;
            }

            if (cLen < c.length) { this.unshift(c.slice(cLen)); }
        });
    });
    global.ipcClient.on('error', function () { process.exit(); });
    global.ipc2Client.on('error', function () { process.exit(); });
}

module.exports = { dispatch: dispatch, connect: connect };
```