dynamicdevices

MethodManager 4 Raw TCP/IP Socket Communication API

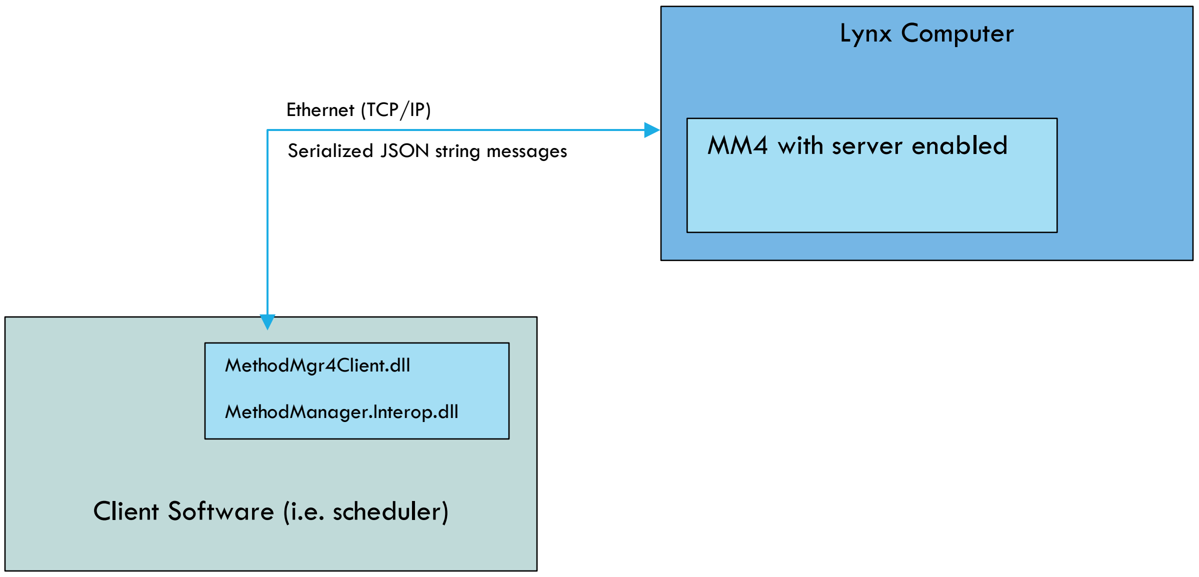
(December 2022)

Contents: **Page:**

MethodManager 4 Raw TCP/IP Socket Communication API

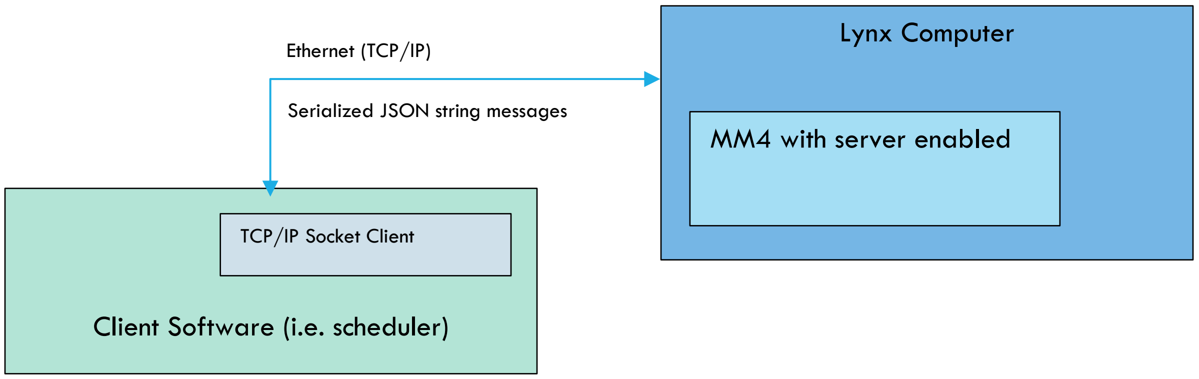
In general, MethodManager 4 (MM4) can be controlled using provided .NET API which included in following files:

- MethodMgr4Client.dll
- MethodManager.Interop.dll

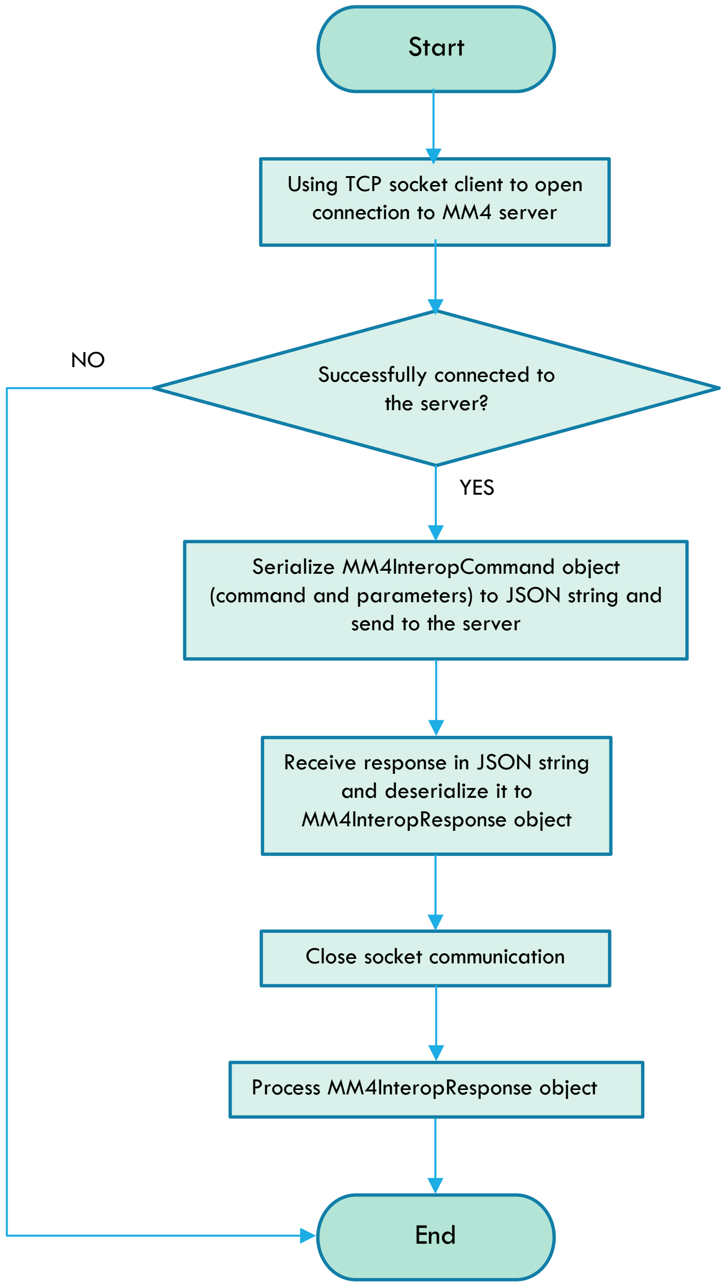

Lynx Computer
Ethernet (TCP/IP)
Serialized JSON string messages
MM4 with server enabled
MethodMgr4Client.dll
MethodManager.Interop.dll
Client Software (i.e. scheduler)

However, the usage of .NET DLL files is limited only to the programming environment that can include and utilize those .NET DLL files.

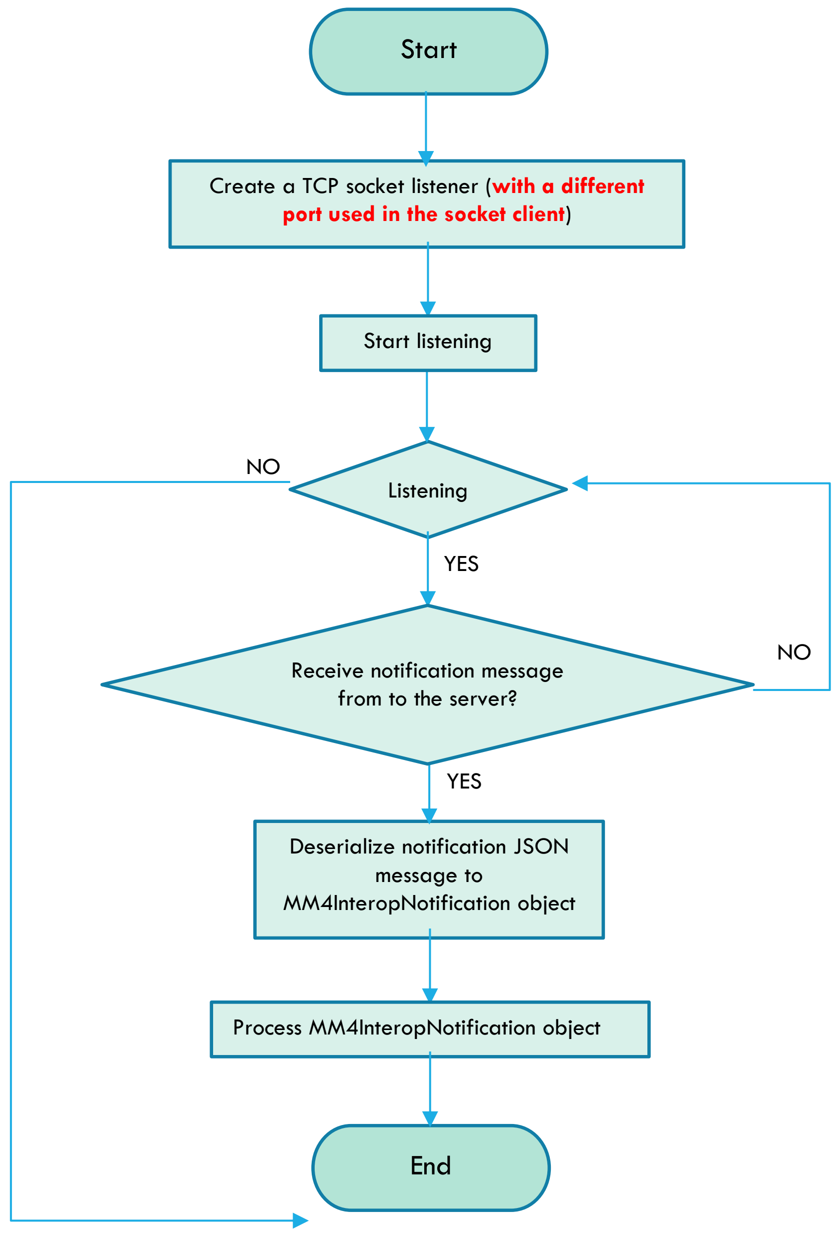
The key purpose of this document is to provide an alternative API information to control MM4 without using MethodMgr4Client.dll and MethodManager.Interop.dll.



Process flowchart for sending MM4InteropCommand and receiving MM4 MM4InteropResponse Objects


Start
Using TCP socket client to open connection to MM4 server
Successfully connected to the server?
NO
YES
Serialize MM4InteropCommand object (command and parameters) to JSON string and send to the server
Receive response in JSON string and deserialize it to MM4InteropResponse object
Close socket communication
Process MM4InteropResponse object
End

Process Flowchart for Receiving Notification Message Events Sent from the MM4 Server


Start
Create a TCP socket listener (**with a different port used in the socket client**)
Start listening
Listening
NO
YES
Receive notification message from to the server?
NO
YES
Deserialize notification JSON message to MM4InteropNotification object
Process MM4InteropNotification object
End

Classes and Enumerations (in C# Language)

```
public class MM4InteropCommand : MM4InteropHeader
{
    public const string WATCH = "WATCH";
    public const string DONT_WATCH = "DONT_WATCH";
    public string ItemName { get; set; }
    public string ItemValue { get; set; }
    public string Password { get; set; }
    public MM4InteropCommand() : base() { }
    public MM4InteropCommand(MM4RemoteCommand commandType) :
        base(commandType)
    {
    }
}

public class MM4InteropResponse : MM4InteropHeader
{
    public const char METHOD_STACK_COMMAND_SEPARATOR = ';';
    public const char METHOD_STACK_PROCESS_SEPARATOR = '|';
    public MM4RemoteError Error { get; set; }
    public MM4RemoteApplicationState ApplicationState { get; set; }
    public string Item { get; set; }
    public MM4RemoteMethodState MethodState { get; set; }
    public MM4RemoteLastMethodResult LastMethodResult { get; set; }
    public string Result { get; set; }
    public MM4InteropResponse() { }
    public MM4InteropResponse(MM4InteropHeader header)
    {
        Command = header.Command;
    }
}

`    public class MM4InteropHeader
{
    public MM4RemoteCommand Command { get; set; }
    protected MM4InteropHeader() { }
    protected MM4InteropHeader(MM4RemoteCommand commandId)
    {
        Command = commandId;
    }
}

public class MM4InteropNotification
{
    public const char VALUE_SEPARATOR = '|';
    public MM4InteropNotificationType NotificationType { get; set; }
    public string ItemName { get; set; }
    public string ItemValue { get; set; }
    public MM4RemoteApplicationState ApplicationState { get; set; }
```

```
}

public enum MM4RemoteCommand
{
    Unknown = 0,
    StartMethod,
    StopMethod,
    GetMethodState,
    GetLastMethodResult,
    GetApplicationState,
    SetVariable,
    GetVariable,
    VariableWatch,
    MethodWatch,
    GetInput,
    QueryWorktablePlate,
    QueryWorktableBarcode,
    QueryWorktableLocation,
    InitializeHardware,
    ConnectHardware,
    ClearErrors
}

public enum MM4RemoteApplicationState
{
    None = 0,
    WorkspaceLoaded = 0x0001,
    SimulationMode = 0x0002,
    MethodRunning = 0x0004,
    MethodPaused = 0x0008,
    MethodErrorPaused = 0x0010,
    ApplicationBlocked = 0x0020,
    EStopEngaged = 0x0040,
    DevicesReady = 0x0080,
    InitializationInProgress = 0x0100
}

public enum MM4RemoteMethodState
{
    Unknown = 0,
    NoActiveMethod,
    Busy,
    Paused,
    ErrorPaused
}

public enum MM4RemoteLastMethodResult
{
    None = 0,
    Success,
        Interrupted,
        Error
    }

    public enum MM4RemoteError
    {
        OK = 0,
        NoWorkspace,
        ApplicationBlocked,
        EStopEngaged,
        PermissionLevelNotUser,
        MethodPermissionLevelNotUser,
        DevicesNotReady,
        NotExecutionMode,
        MethodAlreadyRunning,
        UnknownVariable,
        VariableIsReadOnly,
        UnknownDevice,
        UnknownWorktable,
        UnknownQuery,
        UnknownInput,
        UnknownMethod,
        RemoteControlPswdNotValid,
        NoMethodRunning,
        BadCommandFormat,
        ApplicationError,
        ClientSideError,
        SubMethodOnly
    }

    public enum MM4InteropNotificationType
    {
        Unknown = 0,
        MethodComplete,
        VariableChanged,
        InitializationComplete,
        ConnectionComplete
    }
```

MM4RemoteCommand

Command	**StartMethod (1)**	
Description	Start a specific method	
Parameters	**ItemName**	Method Name
	Password	Username/Password
Response	**Error**	NoWorkspace ApplicationBlocked EStopEngaged PermissionLevelNotUser

		MethodAlreadyRunning NoMethodRunning RemoteControlPswdNotValid UnknownMethod MethodPermissionLevelNotUser ApplicationError	
	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	MM4RemoteError	**Description**
		RemoteControlPswdNotValid	The client at address <Client IP Adderss> did not provide the correct password to start a method.
		UnknownMethod	Method could not be found.
		MethodPermissionLevelNotUser	The requested method cannot be run as the Permission Level 'User'
		ApplicationError	Unknown error/Method execution not available.
	Command	**StartMethod (1)**	

Command	**StopMethod (2)**		
Description	Stop all running method(s)		
Parameter	**Password**	Username/Password	
Response	**Error**	NoWorkspace ApplicationBlocked EStopEngaged PermissionLevelNotUser MethodAlreadyRunning NoMethodRunning RemoteControlPswdNotValid ApplicationError	
	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	**MM4RemoteError**	**Description**
		RemoteControlPswdNotValid	The client at address <Client IP Adderss> did not provide the correct password to stop a method.
	Command	**StopMethod (2)**	

Command	**GetMethodState (3)**		
Description	Query running method(s) state		
Response	**Error**		
	ApplicationState	See MM4RemoteApplicationState	
	Item	Current running main process method name and path	
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	**MethodState**	**Description**

		Unknown NoActiveMethod	
		Busy	In concurrent mode, multiple methods can be run concurrently, concurrent running methods will be stacked with METHOD_STACK_COMMAND_SEPARATOR and METHOD_STACK_PROCESS_SEPARATOR.
		Paused ErrorStopped	Current process name with step index
	Command	**GetMethodState (3)**	

Command	**GetLastMethodResult (4)**		
Description	Query method running result		
Response	**Error**	NoWorkspace MethodAlreadyRunning	
	ApplicationState	See MM4RemoteApplicationState	
	Item	Current running main process method name and path	
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	**LastMethodState**	**Description**
		Error	List of errored method(s)
		Interrupted	List of interrupted method(s)
		Success	Method name
	Command	**GetLastMethodResult (4)**	

Command	**GetApplicationState (5)**	
Description	Query MM4 application state	
Response	**Error**	OK
	ApplicationState	Application State numeric value (as defined in MM4RemoteApplicationState)
	Item	Current workspace name
	MethodState	See MM4RemoteMethodState
	LastMethodResult	See MM4RemoteLastMethodResult
	Result	List of worktable name(s) with comma separated
	Command	**GetApplicationState (5)**

Command	**SetVariable (6)**		
Description	Set variable value		
Parameters	**ItemName**	Variable name	
	ItemValue	Variable value	
	Password	Username/Password	
Response	**Error**	RemoteControlPswdNotValid VariableIsReadOnly NoWorkspace UnknownVariable	
	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	MM4RemoteError	**Description**

		RemoteControlPswdNotValid	The client at address <Client IP Adderss> did not provide the correct password to change a method variable.
	Command	**SetVariable (6)**	

Command	**GetVariable (7)**	
Description	Get variable value	
Parameters	**ItemName**	Variable name
Response	**Error**	NoWorkspace UnknownVariable
	ApplicationState	See MM4RemoteApplicationState
	Item	
	MethodState	See MM4RemoteMethodState
	LastMethodResult	See MM4RemoteLastMethodResult
	Result	Variable value
	Command	**GetVariable (7)**

Command	**VariableWatch (8)**		
Description	Receive notification messages on variable value changed events		
Parameters	**ItemName**	Variable name	
	ItemValue	**WATCH** or **DONT_WATCH** string keyword following by **notification port number**	
Response	**Error**	BadCommandFormat NoWorkspace UnknownVariable	
	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	MM4RemoteError	**Description**
		BadCommandFormat	The notification port command parameter (in ItemValue) could not be resolved.
	Command	**VariableWatch (8)**	

Command	**MethodWatch (9)**		
Description	Receive notification messages on method events		
Parameters	**ItemValue**	**WATCH** or **DONT_WATCH** string keyword following by **notification port number**	
Response	**Error**	BadCommandFormat	
	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	MM4RemoteError	**Description**
		BadCommandFormat	The notification port command parameter (in ItemValue) could not be resolved.
	Command	**MethodWatch (9)**	

Command	GetInput (10)	
Description	Get input value	
Parameters	ItemName	Input name
Response	Error	ApplicationError NoWorkspace UnknownInput
	ApplicationState	See MM4RemoteApplicationState
	Item	
	MethodState	See MM4RemoteMethodState
	LastMethodResult	See MM4RemoteLastMethodResult
	Result	Input state string (**Active** or **Inactive**) MM4RemoteError — **Description** ApplicationError — Failed input read: <selected input>
	Command	**GetInput (10)**

Command	QueryWorktablePlate (11)	
Description	Query worktable plate	
Parameters	ItemName	Worktable name
	ItemValue	Plate name
Response	Error	NoWorkspace NoMethodRunning UnknownWorktable MethodAlreadyRunning ApplicationError
	ApplicationState	See MM4RemoteApplicationState
	Item	
	MethodState	See MM4RemoteMethodState
	LastMethodResult	See MM4RemoteLastMethodResult
	Result	Formatted result string **MM4RemoteError** — **Description** ApplicationError — Error during worktable lookup. Error during query: + exception message
	Command	**QueryWorktablePlate (11)**

Command	QueryWorktableBarcode (12)	
Description	Query worktable barcode	
Parameters	ItemName	Worktable name
	ItemValue	Plate barcode
Response	Error	NoWorkspace NoMethodRunning UnknownWorktable MethodAlreadyRunning ApplicationError
	ApplicationState	See MM4RemoteApplicationState
	Item	
	MethodState	See MM4RemoteMethodState
	LastMethodResult	See MM4RemoteLastMethodResult
	Result	Formatted result string **MM4RemoteError** — **Description** ApplicationError — Error during worktable lookup.

			Error during query: + exception message
	Command	**QueryWorktableBarcode (12)**	
Command	**QueryWorktableLocation (13)**		
Description	Query worktable location		
Parameters	**ItemName**	Worktable name	
	ItemValue	Location name	
Response	**Error**	NoWorkspace NoMethodRunning UnknownWorktable MethodAlreadyRunning ApplicationError	
	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	Formatted result string	
		MM4RemoteError	**Description**
		ApplicationError	Error during worktable lookup. Error during query: + exception message
	Command	**QueryWorktableLocation (13)**	

Command	**InitializeHardware (14)**		
Description	Initialize hardware		
Parameters	**Password**	Username/Password	
	ItemValue	Notification port number	
Response	**Error**	BadCommandFormat RemoteControlPswdNotValid ApplicationError	
	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	Formatted result string	
		MM4RemoteError	**Description**
		BadCommandFormat	The notification port command parameter (in ItemValue) could not be resolved.
		RemoteControlPswdNotValid	The client at address <Client IP Adderss> did not provide the correct password to initialize hardware.
		ApplicationError	TCP server unable to initiate hardware initialization. Method execution not available.
	Command	**InitializeHardware (14)**	

Command	**ClearErrors (15)**	
Description	Query method running result	
Parameters	**Password**	Username/Password
Response	**Error**	BadCommandFormat RemoteControlPswdNotValid ApplicationError

	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	Formatted result string	
		MM4RemoteError	**Description**
		BadCommandFormat	The notification port command parameter (in ItemValue) could not be resolved.
		RemoteControlPswdNotValid	The client at address <Client IP Adderss> did not provide the correct password to initialize hardware.
		ApplicationError	TCP server unable to initiate hardware initialization. Method execution not available.
	Command	**ClearErrors (15)**	

Command	**ConnectHardware (16)**		
Description	Initialize hardware		
Parameters	Password	Username/Password	
	ItemValue	Notification port number	
Response	Error	BadCommandFormat RemoteControlPswdNotValid ApplicationError	
	ApplicationState	See MM4RemoteApplicationState	
	Item		
	MethodState	See MM4RemoteMethodState	
	LastMethodResult	See MM4RemoteLastMethodResult	
	Result	Formatted result string	
		MM4RemoteError	**Description**
		BadCommandFormat	The notification port command parameter (in ItemValue) could not be resolved.
		RemoteControlPswdNotValid	The client at address <Client IP Adderss> did not provide the correct password to connect hardware.
		ApplicationError	TCP server unable to initiate hardware initialization. Method execution not available.
	Command	**ConnectHardware (16)**	

MM4InteropNotification

NotificationType	**MethodComplete (1)**
ItemName	Method name
ItemValue	“ERROR” if error(s) occurred, otherwise empty
MM4RemoteApplicationState	See MM4RemoteApplicationState

NotificationType	**VariableChanged (2)**
ItemName	Variable name

ItemValue	Variable value
MM4RemoteApplicationState	See MM4RemoteApplicationState

NotificationType	**InitializationComplete (3)**
ItemName	"ERROR" if error(s) occurred, otherwise empty
ItemValue	Error information, otherwise empty
MM4RemoteApplicationState	See MM4RemoteApplicationState

NotificationType	**ConnectionComplete (4)**
ItemName	"ERROR" if error(s) occurred, otherwise empty
ItemValue	Error information, otherwise empty
MM4RemoteApplicationState	See MM4RemoteApplicationState

MM4RemoteError

RemoteError	Value	Description
OK	0	The server understood and executed the command without error.
NoWorkspace	1	The server does not have a Workspace loaded.
ApplicationBlocked	2	The application is performing a task that prevents the server from responding correctly.
EStopEngaged	3	The Emergency-Stop is engaged on the server and no commands can be performed.
PermissionLevelNotUser	4	The current user level on the server is other than 'User', which is required to permit remote commands.
MethodPermissionLevelNotUser	5	A method start was requested but the method does not have the 'User Permission Level' flag selected.
DevicesNotReady	6	The Devices in the Workspace are not initialized and no methods can be performed.
NotExecutionMode	7	The server is in Test mode and cannot execute a command.
MethodAlreadyRunning	8	A method is currently running and a new method cannot be started or last method results cannot be reported.
UnknownVariable	9	A method variable name was not recognized on the server.
VariableIsReadOnly	10	An attempt was made to write to a read-only method variable on the remote server.
UnknownDevice	11	A client command referenced a Device that that the server does not recognize.
UnknownWorktable	12	A client command referenced a Worktable that that the server does not recognize.
UnknownQuery	13	A client command requested a query that the server does not recognize.
UnknownInput	14	A client command requested the state of an input that the server does not recognize.
UnknownMethod	15	A client command referenced a Worktable that that the server does not recognize.
RemoteControlPswdNotValid	16	A client issues a command using an incorrect Remote Password.
NoMethodRunning	17	A client requested a stop method command but no method was running.
BadCommandFormat	18	A command format was not recognized.

ApplicationError	19	The server encountered an error that prevented a correct response.
ClientSideError	20	The client was unable to establish a connection to the server.
SubMethodOnly	21	The method is flagged to only ever run as a sub-method.

MM4RemoteMethodState

RemoteMethodState	Value	Description
Unknown	0	Unknown state.
NoActiveMethod	1	There is no active method.
Busy	2	A method is executing.
Paused	3	A method is paused due to a non-error condition.
ErrorPaused	4	A method is paused due to an error condition.

MM4RemoteLastMethodResult

RemoteLastMethodResult	Value	Description
None	0	There is no record of a previous method.
Success	1	The last method executed completed without error.
Interrupted	2	A non-error interruption that prevented completion. Can be for a variety of reasons.
Error	3	The last method executed terminated due to an error.

MM4RemoteLastMethodResult

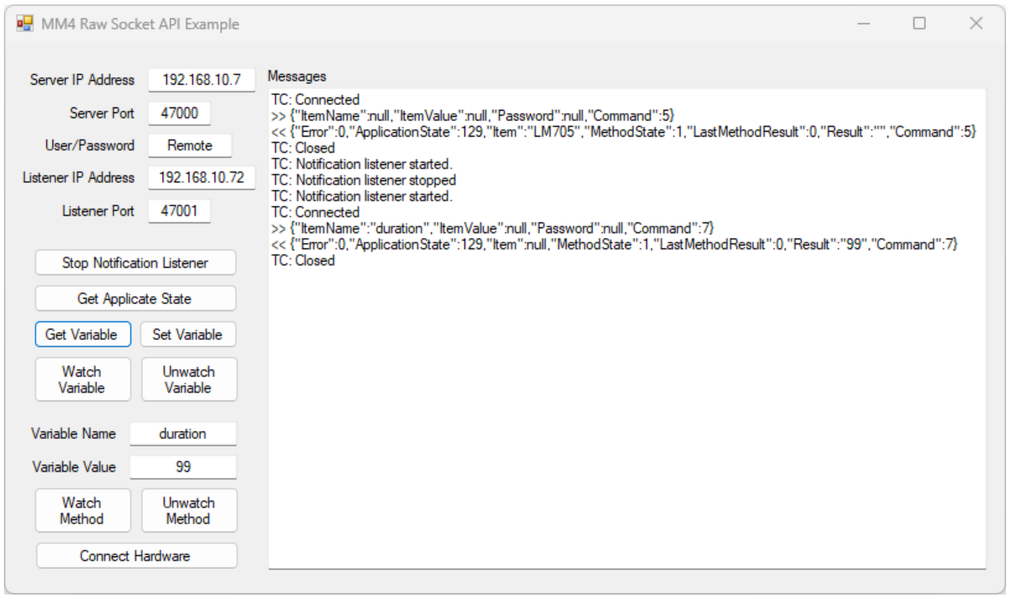
RemoteLastMethodResult	Value	Description
None	0	None
WorkspaceLoaded	1 (0x0001)	A workspace is loaded.
SimulationMode	2 (0x0002)	The application is in test mode, not execute mode.
MethodRunning	4 (0x0004)	The method(s) is active, but may also be in a paused state waiting for user intervention.
MethodPaused	8 (0x0008)	The method(s) is paused due to a non-error state.
MethodErrorPaused	16 (0x0010)	The method(s) is paused due to an error state.
ApplicationBlocked	32 (0x0020)	The application is blocked waiting for user interaction.
EStopEngaged	64 (0x0040)	The Emergency Stop is engaged.
DevicesReady	128 (0x0080)	All Workspace device are ready to perform method commands.
InitializationInProgress	256 (0x0100)	The device initialization/connection process is ongoing.

MM4InteropNotificationType

InteropNotificationType	Value	Description
Unknown	0	Unknown (should never happen).
MethodComplete	1	This notification occurs when a method is complete.
VariableChanged	2	This notification occurs when monitored variable(s) value changed.
InitializationComplete	3	This notification occurs when system initialization is complete.
ConnectionComplete	4	This notification occurs when system connection is complete.

Example Source Codes

This example application is written in C# to demonstrate the MM4's raw TCP/IP socket communication API. The Visual Studio project source code is available at https://github.com/DynamicDevicesInc/RawSocketAPI



Server IP Address	192.168.10.7
Server Port	47000
User/Password	Remote
Listener IP Address	192.168.10.72
Listener Port	47001
Variable Name	duration
Variable Value	99

Messages

```
TC: Connected
>> {"ItemName":null,"ItemValue":null,"Password":null,"Command":5}
<< {"Error":0,"ApplicationState":129,"Item":"LM705","MethodState":1,"LastMethodResult":0,"Result":"","Command":5}
TC: Closed
TC: Notification listener started.
TC: Notification listener stopped
TC: Notification listener started.
TC: Connected
>> {"ItemName":"duration","ItemValue":null,"Password":null,"Command":7}
<< {"Error":0,"ApplicationState":129,"Item":null,"MethodState":1,"LastMethodResult":0,"Result":"99","Command":7}
TC: Closed
```

Create and Process MM4InteropCommand Transactions

```
private bool ProcessTransaction(MM4InteropCommand cmd)
{
    _lastErrorMsg = "";
    _response = new MM4InteropResponse();
    _lastError = _response.Error = MM4RemoteError.ClientSideError;
    if (_tcpClient == null)
    {
        _lastErrorMsg = "Client not active. Call Initialize() with appropriate IP addresses.";
        return false;
    }
    string err = "";
    if (!CommandTransaction(cmd, ref _response, out err))
    {
        _lastErrorMsg = "Client-side transaction failure: " + err;
        return false;
    }
    _lastError = _response.Error;
    if (_response.Error != MM4RemoteError.OK)
    {
        _lastErrorMsg = _response.Error.ToString() +
            (!string.IsNullOrEmpty(_response.Result) ? (", Hint: " + _response.Result) : "");
        return false;
    }
```

```
        return true;
    }

    public bool CommandTransaction(MM4InteropCommand cmd, ref MM4InteropResponse response, out string errorMsg)
    {
        errorMsg = "";
        IAsyncResult result = null;
        try
        {
            // String to store the response ASCII representation.
            String responseData = String.Empty;

            _serverAddr = IPAddress.Parse(txtIPAddress.Text);
            _serverPort = Convert.ToUInt16(txtPort.Text);

            using (TcpClient client = new TcpClient())
            {
                // -- Try Asynch --
                result = client.BeginConnect(_serverAddr, _serverPort, null, null);

                // Small delay
                Thread.Sleep(100);

                if (!result.AsyncWaitHandle.WaitOne(3000))
                {
                    LogMessage("TC: connection timeout");
                    // Close the socket and bail.
                    client.Client.Close();
                    client.Close();
                    result.AsyncWaitHandle.Close();
                    result.AsyncWaitHandle.Dispose();
                    result = null;
                    LogMessage("TC: Connection cleanup");
                    throw new Exception("Server connection timeout");
                }

                client.EndConnect(result);
                LogMessage("TC: Connected");

                // Translate the passed message into ASCII and store it as a Byte array.
                string dataString = JsonConvert.SerializeObject(cmd);
                Byte[] data = System.Text.Encoding.ASCII.GetBytes(dataString);

                // Get a client stream for reading and writing.
                using (NetworkStream stream = client.GetStream())
                {
                    // Send the message to the connected TcpServer.
                    LogMessage(">> " + dataString);
                    stream.Write(data, 0, data.Length);

                    // Receive the TcpServer.response.
                    // Buffer to store the response bytes.
                    data = new Byte[BUFF_SZ];

                    // Read the first batch of the TcpServer response bytes.
                    Int32 bytes = stream.Read(data, 0, data.Length);
                    responseData = Encoding.ASCII.GetString(data, 0, bytes).Trim();

                    LogMessage("<< " + responseData);
                    response = JsonConvert.DeserializeObject<MM4InteropResponse>(responseData);
                    // Close everything.
                    stream.Close();
                }
                client.Client.Close();
                client.Close();

                LogMessage("TC: Closed");
            }
            // The JSON deserialization might fail.
            if (response == null)
            {
                LogMessage("TC: Response data not deserialized: " + responseData);
            }
            return (response != null);
            }
            catch (Exception ex)
            {
                errorMsg = ex.Message;
                LogMessage("TC: AsyncWaitHandle exception: " + errorMsg);
                if (ex.InnerException != null)
                {
                    LogMessage("TC: inner exception: " + ex.InnerException.Message);
                }

                if (result != null)
                {
                    result.AsyncWaitHandle.Close();
                    result.AsyncWaitHandle.Dispose();
                    result = null;
                }
            }
            return false;
        }

        public MM4RemoteError StartMethod(string methodName)
        {
            ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.StartMethod)
            {
                ItemName = methodName,
                Password = txtUserOrPassword.Text
            });
            return _lastError;
        }

        public MM4RemoteError StopMethod()
        {
            ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.StopMethod)
            {
                Password = txtUserOrPassword.Text
            });
            return _lastError;
        }

        public MM4RemoteError GetMethodState(out string activeMethodName, out MM4RemoteMethodState methodState)
        {
            activeMethodName = "";
            methodState = MM4RemoteMethodState.Unknown;
            if (ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.GetMethodState)))
            {
                activeMethodName = _response.Result;
                methodState = _response.MethodState;
            }
            return _lastError;
        }

        public MM4RemoteError GetLastMethodResult(out string lastMethodName, out MM4RemoteLastMethodResult lastMethodResult)
        {
            lastMethodName = "";
            lastMethodResult = MM4RemoteLastMethodResult.None;
            if (ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.GetLastMethodResult)))
            {
                lastMethodName = _response.Result;
                lastMethodResult = _response.LastMethodResult;
            }
            return _lastError;
        }

        public MM4RemoteError GetApplicationState(out MM4RemoteApplicationState applicationState, out string workspaceName, out string
    worktableNames)
        {
            applicationState = MM4RemoteApplicationState.None;
            workspaceName = "";
            worktableNames = "";
            if (ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.GetApplicationState)))
            {
                applicationState = _response.ApplicationState;
                workspaceName = _response.Item;
                worktableNames = _response.Result;
        }
        return _lastError;
    }

    public MM4RemoteError SetVariable(string variableName, string value)
    {
        ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.SetVariable)
        {
            ItemName = variableName,
            ItemValue = value,
            Password = txtUserOrPassword.Text
        });
        return _lastError;
    }

    public MM4RemoteError GetVariable(string variableName, out string value)
    {
        value = "";
        if (ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.GetVariable)
        {
            ItemName = variableName
        }))
        {
            value = _response.Result;
        }
        return _lastError;
    }

    public MM4RemoteError VariableWatch(string variableName, bool watch)
    {
        if (_notificationPort == 0)
        {
            _lastErrorMsg = "Watch is not allowed because there is no valid client notification port.";
            return MM4RemoteError.ClientSideError;
        }
        ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.VariableWatch)
        {
            ItemName = variableName,
            ItemValue = (watch ? MM4InteropCommand.WATCH : MM4InteropCommand.DONT_WATCH) + _notificationPort.ToString()
        });
        return _lastError;
    }

    public MM4RemoteError MethodWatch(bool watch)
    {
        if (_notificationPort == 0)
        {
            _lastErrorMsg = "Watch is not allowed because there is no valid client notification port.";
            return MM4RemoteError.ClientSideError;
        }
        ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.MethodWatch)
        {
            ItemValue = (watch ? MM4InteropCommand.WATCH : MM4InteropCommand.DONT_WATCH) + _notificationPort.ToString()
        });
        return _lastError;
    }

    public MM4RemoteError GetInput(string inputName, out bool active)
    {
        active = false;
        if (ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.GetInput)
        {
            ItemName = inputName
        }))
        {
            active = bool.Parse(_response.Result);
        }
        return _lastError;
    }

    public MM4RemoteError QueryWorktablePlate(string worktableFullName, string plateName, out string queryResults)
    {
        queryResults = "";
        if (ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.QueryWorktablePlate)
        {
            ItemName = worktableFullName,
            ItemValue = plateName
        }))
        {
            queryResults = _response.Result;
        }
        return _lastError;
    }
    public MM4RemoteError QueryWorktableBarcode(string worktableFullName, string barCode, out string queryResults)
    {
        queryResults = "";
        if (ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.QueryWorktablePlate)
        {
            ItemName = worktableFullName,
            ItemValue = barCode
        }))
        {
            queryResults = _response.Result;
        }
        return _lastError;
    }

    public MM4RemoteError QueryWorktableLocation(string worktableFullName, string locationName, out string queryResults)
    {
        queryResults = "";
        if (ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.QueryWorktableLocation)
        {
            ItemName = worktableFullName,
            ItemValue = locationName
        }))
        {
            queryResults = _response.Result;
        }
        return _lastError;
    }

    public MM4RemoteError InitializeHardware()
    {
        ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.InitializeHardware)
        {
            Password = txtUserOrPassword.Text,
            ItemValue = _notificationPort.ToString()
        });
        return _lastError;
    }

    public MM4RemoteError ClearErrors()
    {
        ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.ClearErrors)
        {
            Password = txtUserOrPassword.Text
        });
        return _lastError;
    }

    public MM4RemoteError ConnectHardware()
    {
        ProcessTransaction(new MM4InteropCommand(MM4RemoteCommand.ConnectHardware)
        {
            Password = txtUserOrPassword.Text,
            ItemValue = _notificationPort.ToString()
        });
        return _lastError;
    }
```

Start, Stop and Process MM4InteropNotification

```
public void StartNotificationListener()
{
    _clientAddr = IPAddress.Parse(txtListenerIPAddress.Text);
    _notificationPort = Convert.ToUInt16(txtListenerPort.Text);

    if (_clientAddr == IPAddress.None)
    {
        return;
    }
    (new Thread(() =>
    {
        string errMsg = "";
        try
        {
            // TcpListener server = new TcpListener(port);
            _listener = new TcpListener(_clientAddr, _notificationPort);

            // Start listening for client requests.
            _listener.Start();
            LogMessage("TC: Notification listener started.");

            // Buffer for reading data
            Byte[] bytes = new Byte[BUFF_SZ];
            _listening = true;
            // Enter the listening loop.

            SetControlText(btnNotificationListenerControl, "Stop Notification Listener");
            while (Listening)
            {
                // Perform a blocking call to accept requests.
                // You could also user server.AcceptSocket() here.
                using (TcpClient client = _listener.AcceptTcpClient())
                {
                    LogMessage("TC: Notification.");

                    // Get a stream object for reading and writing
                    int i = 0;
                    using (NetworkStream stream = client.GetStream())
                    {
                        i = stream.Read(bytes, 0, bytes.Length);
                        stream.Close();
                    }
                    // Shutdown and end connection
                    client.Client.Close();
                    client.Close();
                    if (i != 0)
                    {
                        // Translate data bytes to a ASCII string.
                        ProcessNotification(Encoding.ASCII.GetString(bytes, 0, i));
                    }
                    // Old location // Shutdown and end connection
                    // Old location client.Close();
                    LogMessage("TC: Notification processed.");
                }
            }
        }
        catch (SocketException ex)
        {
            if (ex.SocketErrorCode != SocketError.Interrupted)
            {
                errMsg = ex.Message;
            }
        }
        catch (Exception ex)
        {
            errMsg = ex.Message;
        }
        finally
        {
            // Stop listening for new clients.
            if (_listener != null)
            {
                _listener.Stop();
            }
        }
        _listening = false;
        _listener = null;
        if (!string.IsNullOrEmpty(errMsg))
        {
            LogMessage("TC: Notification listener failed: " + errMsg);
        }
    }) { Name = "Notification Listener" }).Start();
}

public void StopNotificationListener()
{
    if (_listening)
    {
        _listening = false;
        if (_listener != null)
        {
            _listener.Stop();
        }
        Thread.Sleep(100);
        LogMessage("TC: Notification listener stopped");
    }

    SetControlText(btnNotificationListenerControl, "Start Notification Listener");
}

private void ProcessNotification(string message)
{
    LogMessage("Client ProcessNotification: " + message);

    if ((message.Replace('{', '\0').Length - message.Replace('}', '\0').Length) == 0)
    {
        try
        {
            MM4InteropNotification notification = JsonConvert.DeserializeObject<MM4InteropNotification>(message.Trim());

            if (notification.NotificationType == MM4InteropNotificationType.MethodComplete)
            {
                LogMessage("NotificationType: MethodComplete");
                if (!string.IsNullOrEmpty(notification.ItemName))
                    LogMessage("ItemName: " + notification.ItemName);
                if (!string.IsNullOrEmpty(notification.ItemValue))
                    LogMessage("ItemValue: " + notification.ItemValue);
            }
            else if (notification.NotificationType == MM4InteropNotificationType.InitializationComplete)
            {
                LogMessage("NotificationType: InitializationComplete");
                if (!string.IsNullOrEmpty(notification.ItemName))
                    LogMessage("ItemName: " + notification.ItemName);
                LogMessage("ItemValue: " + notification.ItemValue);
            }
            else if (notification.NotificationType == MM4InteropNotificationType.ConnectionComplete)
            {
                LogMessage("NotificationType: ConnectionComplete");
                if (!string.IsNullOrEmpty(notification.ItemName))
                    LogMessage("ItemName: " + notification.ItemName);
                if (!string.IsNullOrEmpty(notification.ItemValue))
                    LogMessage("ItemValue: " + notification.ItemValue);
            }
            else if (notification.NotificationType == MM4InteropNotificationType.VariableChanged)
            {
                LogMessage("NotificationType: ConnectionComplete");
                if (!string.IsNullOrEmpty(notification.ItemName))
                    LogMessage("ItemName: " + notification.ItemName);
                if (!string.IsNullOrEmpty(notification.ItemValue))
                    LogMessage("ItemValue: " + notification.ItemValue);
            }
        }
        catch { }
    }
}
```