MM4 Remote Monitoring And Control v1.1
MethodManager4 ver 1.2.x

**Server Configuration**

Method Manager 4 (MM4) may be configured to accept instructions and provide status updates to other applications via a TCP server. A client application can monitor the state of the method being executed, get the state of the MM4 application, get method variable values, querying the plates on the worktables, and get the status of a inputs. If the client application can supply the valid remote control password that client can also start methods, stop methods, change the value of variables, and initiate the hardware initialization process. The client can also request asynchronous notification when a specific variable changes state, a method completed, or the initialization sequence completes. Any number of client applications can monitor the MM4 processes, but only clients that supply the remote control password can perform the control functions.

The remote clients can only issues commands to MM4 when MM4 is in the 'User' Permission Level (see document "*MM4 Login and Permission Levels.PDF*") and then only if the client includes the remote control password in the command instructions.

Method Manager 4 Configuration settings for remote client connections are set in the server configuration file (C:\ProgramData\MethodManager4\methodmanager.server.config) as follows:

- **TcpServerEnabled**: Must be set to "True" in order to enable the TCP server.
- **TcpServerAddress**: Specify a valid local IP address for the TCP server.
- **TcpServerPort**: This must be a valid free port number on the computer running MM4. If left blank the default value of 47000 will be used.

## Client API Implementation

The remote client communicates with MM4 over a TCP connection using the class MM4Client in MethodMgr4Client.DLL. This class can be implemented either as a .NET library or a COM object. Note that network permissions must be provided for the TCP transactions on both endpoint machines.

Most of the methods in the API return an instance of **`MM4RemoteError`** to indicate success or the cause of the command or request failure. The exception to this is the **`Connect()`** function which returns true on success and if there is a connection failure a verbose error message is placed in the **`connectionError`** parameter. Some commands can only be performed if MethodManager4 is in the correct state to process the command. To determine the state of the application use **`GetApplicationState()`** and to determine the state of a currently running method use **`GetMethodState()`**.

## MM4Client API Methods

**`bool Connect(string serverAddr, int serverPort, string clientAddr, int notificationPort, string password, out string connectionError)`**

Initializes the endpoints for TCP transactions and established a connection with an instance of MethodManager4. This must be called before any communication with MM4 is performed.

**Return:**

True if successful, otherwise false. The parameter **`connectionError`** will contain a description of the reason for the connection failure.

**Parameters:**

**`serverAddr`** [in]: IP address of the MethodManager4 instance to be accessed.

**`serverPort`** [in]: IP port of the MethodManager4 instance to be accessed.
**`clientAddr`** [in]: IP address of local client instance.
**`notificationPort`** [in]: IP port of local client instance to receive notifications.
**`password`** [in]: Remote control password.
**`connectionError`**[out]: Error message indicating reason for connection failure.

**Notes:** The client initiated transactions will use an internally specified port, however all notification from the server will go to the client port specified by **`notificationPort`**. The remote control password is only required for a client to initiate a change in the MM4 server, but is not required to poll the server status or subscribe to notification events.

**`void Disconnect()`**

Closes the TCP communications. This should be called before exiting any application that implements MM4Client.

**Return:**

None.

**Out variables:**

None.

**`MM4RemoteError GetLastError(out string lastError)`**

Places a description of the error associated with the last command in the parameter **`lastError`**.

**Return:**

The error code from the last command (see table below).

**Parameters:**

**`lastError`** [out]: Description of the last error. Depending on the error this may have additional information specific to the type of error.

**`MM4RemoteError StartMethod(string methodName)`**

Attempts to start a method. The method name must not include the file extension and the method must be valid for execution by the Permission Level of 'User'. If the method is not already loaded in MM4 it will first be loaded prior to execution.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Parameters:**

**`methodName`** [in]: Name of method, not including file extension.

**Notes:** There are a number of reasons a method may not start. Before calling this function first check the state of the MM4 using **`GetApplicationState()`** and confirm the application state is only WorkspaceLoaded and DevicesReady.

**`MM4RemoteError StopMethod()`**

Stops an active method using the same termination options that are specified for the Permission Level of 'User' in the MM4 instance.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Parameters:**

None.

**Notes:** This will terminate the method and require a user to intervene to clear the error in MethodManager4. This should only be used if the client detects an unrecoverable process error and needs to halt the entire application.

**`MM4RemoteError GetMethodState(out string activeMethodName, out MM4RemoteMethodState methodState)`**

Provides the current state of a method currently being executed in the MM4 instance.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Parameters:**

**`activeMethodName`** [out]: The method call stack for currently executing methods.

**`methodState`** [out]: The state of the active method method.

**Notes:**

The call stack lists the names of sub-methods in the reverse of the call hierarchy. Each method in the call stack is identified by a comma-separated list of its method name, the line number being executed, and the name of the command. Each method entry is separated by a semicolon, and in the case of multiple method threads executing in parallel the separate methods stacks will be delineated using the pipe character ('|').

**`MM4RemoteError GetApplicationState(out MM4RemoteApplicationState applicationState, out string workspaceName, out string worktableFullNames)`**

Retrieves information about the application state, the currently loaded Workspace, and the Worktables in the Workspace.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Parameters:**

**`applicationState`** [out]: The current state of the application (see Table 1).

**`workspaceName`** [out]: The name of the currently loaded Workspace if any, otherwise blank.

**`worktableFullNames`** [out]: All Worktables in the currently loaded Workspace if any, otherwise blank.

**Notes:**

If no workspace is loaded then both **`workspaceName`** and **`worktableFullNames`** will be blank. If there is a current Workspace then **`worktableFullNames`** will be a comma-seperated list of compound Worktable names. A compound worktable name has two parts, the name of the parent device and the name of the worktable under that parent device separated by a ':' character. Thus a workspace with two liquid handlers, each with a worktable called “Left” would return in **`worktableFullNames`** the string “Lynx1:Left,Lynx2:Left”.

**`MM4RemoteError SetVariable(string variableName, string value)`**

Sets the value of a method variable.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Parameters:**

**`variableName`** [in]: The name of the variable to set.

**`value`** [in]: The new value of the variable.

**`MM4RemoteError GetVariable(string variableName, out string value)`**

Gets the value of a method variable.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Parameters:**

**variableName** [in]: The name of the variable to set.
**value** [out]: The current value of the variable.

**MM4RemoteError VariableWatch(string variableName, bool watch)**
Registers or un-registers the client to be notified when a method variable is written to in the MM4 application.
**Return:**
The error code if the command was not successful, or **MM4RemoteError.OK** if successful.
**Parameters:**
**variableName** [in]: The name of the variable to register or un-registers notifications.
**watch** [in]: A flag to set the notification on (true) or off (false).
**Notes:**
The client application must also subscribe to the event notification using **OnVariableChanged()** order to receive change notifications for the specified variable. This client function can be used in combination with the MM4 command 'Wait On Variable' to synchronize method execution with external integration applications.

**MM4RemoteError MethodWatch(bool watch)**
Registers or un-registers the client to be notified when a method or sub-method completes.
**Return:**
The error code if the command was not successful, or **MM4RemoteError.OK** if successful.
**Parameters:**
**watch** [in]: A flag to set the notification on (true) or off (false).
**Notes:**
The client application must also subscribe to the event notification using **OnMethodChanged()** in order to be notified of method completion.

**MM4RemoteError GetInput(string inputName, out bool active)**
Test the state of a digital input.
**Return:**
The error code if the command was not successful, or **MM4RemoteError.OK** if successful.
**Parameters:**
**inputName** [in]: The full name of the input as shown in MM4.
**active** [out]: A flag indicating the state of the input. True indicates the input is active and false indicates the input is inactive.

**MM4RemoteError QueryWorktablePlate(string worktableFullName, string plateName, out string queryResults)**
Performs a query on the specified worktable for the plate with the specified name.
**Return:**
The error code if the command was not successful, or **MM4RemoteError.OK** if successful.
**Parameters:**
**worktableFullName** [in]: Name of the worktable to perform the search on.
**plateName** [in]: Plate name to search for.
**queryResults** [out]: The worktable location of the plate if found.
**Notes:**
The format of **queryResults** is a comma-separated string. The first value is the number of matching plates found followed by entries for each plate. The plate entries are semicolon-separated values as follows:

plate name, plate bar code, worktable location, 0-based location in a plate stack at that location, and a flag indicating if the plate is top-most in the stack. If no plate is found then the result is a string with the value "0".

**`MM4RemoteError QueryWorktableBarcode(string worktableFullName, string barCode, out string queryResults)`**

Performs a query on the specified worktable for the plate with the specified bar code.

**Parameters:**

**`worktableFullName`** [in]: Name of the worktable to perform the search on.

**`barCode`** [in]: Plate bar code to search for.

**`queryResults`** [out]: The worktable location of the plate if found.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Notes:**

See the description of **`QueryWorktablePlate`** for a description of the results in **`queryResults`**.

**`MM4RemoteError QueryWorktableLocation(string worktableFullName, string locationName, out string queryResults)`**

Performs a query on the specified worktable for all plates in the location with the specified name.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Parameters:**

**`worktableFullName`** [in]: Name of the worktable to perform the search on.

**`locationName`** [in]: Location to retrieve the plate stack contents.

**`queryResults`** [out]: The worktable location if the plate is found, otherwise and empty string.

**Notes:**

See the description of **`QueryWorktablePlate`** for a description of the results in **`queryResults`**.

**`MM4RemoteError InitializeHardware()`**

Starts a hardware initialization in the MM4 application using the default hardware initialization selections.

**Return:**

The error code if the command was not successful, or **`MM4RemoteError.OK`** if successful.

**Parameters:**

None.

**Notes:**

This method returns immediately with an error code indicating whether the hardware initialization process was started successfully or not. Hardware initialization can be a lengthy process and completion of it can be detected either by monitoring the application state flag **`InitializationInProgress`** or by subscribing to the event notification using **`OnInitializationComplete()`**.

## API Notification Events

**`void OnMethodComplete(string methodName, bool error, MM4RemoteApplicationState applicationState)`**

Invoked when a method or sub-method completes.

**Parameters:**

**`methodName`** [in]: Name of the completed method or sub-method.

**error** [in]: True if the method or sub-method completed due to an error condition.
**applicationState** [in]: The current state of the application (see Table 1).

**void OnVariableChanged(string variableName, string value, MM4RemoteApplicationState applicationState)**

Invoked when a method variable is written to.

**Parameters:**

**variableName** [in]: Name of the variable.
**value** [in]: Value of the variable.
**applicationState** [in]: The current state of the application (see Table 1).

**void OnInitializationComplete(bool error, string errorMessages, MM4RemoteApplicationState applicationState)**

Invoked when a hardware initialization process that was requested by this instance of the MM4Client class has finished.

**Parameters:**

**error** [in]: True if the hardware initialization encountered an error condition.
**errorMessages** [in]: A list of any error messages generated during the hardware initialization. Messages within the string are separated using the '|' character.
**applicationState** [in]: The current state of the application (see Table 1).

## API Constants

### Table 1, MM4RemoteApplicationState

The application state is a combination of flags.

```
None = 0
WorkspaceLoaded = 1
SimulationMode = 2
MethodRunning = 4
MethodPaused = 8
MethodStopped = 16
ApplicationBlocked = 32
EStopEngaged = 64
DevicesReady = 128
InitializationInProgress = 256
```

### Table 2, MM4RemoteMethodState

The MM4RemoteMethodState is a value indicating the state of the currently executing method, if any.

```
Unknown = 0
NoActiveMethod = 1
Busy = 2
Paused = 3
Stopped = 4
Error = 5
Completed = 6
```

**Table 3, MM4RemoteError**

The MM4RemoteError indicates the reason a remote command failed. It includes failures that originated on the client side, errors in the MM4 application, and errors related to an invalid method or application state.

```
OK = 0
NoWorkspace = 1
ApplicationBlocked = 2
EStopEngaged = 3
PermissionLevelNotUser = 4
MethodPermissionLevelNotUser = 5
DevicesNotReady = 6
NotExecutionMode = 7
MethodAlreadyRunning = 8
UnknownVariable = 9
VariableIsReadOnly = 10
UnknownDevice = 11
UnknownWorktable = 12
UnknownQuery = 13
UnknownInput = 14
UnknownMethod = 15
RemoteControlPswdNotValid = 16
NoMethodRunning = 17
BadCommandFormat = 18
ApplicationError = 19
ClientSideError = 20
```