

 Home

 Next





Mach4 Core API

Build 3797

Authors: Brian Barker, Steve Murphree
©Copyright: Newfangled Solutions™ 2013-2018
Rev:2.01
2018-05-14

Contents

- [Getting Started](#)
- [API Error Return Codes](#)
- [API Reference by Category](#)
- [Alphabetical API Reference](#)
- [Plugin Development](#)

Getting Started

- [Supported Platforms](#)
- [Compiling](#)
- [Linking](#)



Home



Up a level



Previous



Next

Supported Platforms

At this time the only supported platforms are Windows, Linux, Mac. The only plugins that will be supported for both Windows, Linux and Mac will need to be programmed in C++ or C.

Windows:

Plugins can be written in any language that supports interfacing to a flat C API interface. Tests have been done with C++, C# and VB. Other languages such as Action Script, Delphi, Python and etc could be supported but are not tested due to the limited use in the windows world.

Linux:

GUIs need to be written in C or C++ with wxWidgets if they are to have a toolpath display. No testing has been conducted to see if other languages (wyPython, wxPerl...) with wxWidgets will work. Any language with wx should work but it is out of the scope of this help manual to assist with that process.

Mac:

More work needs to be done on this platform :)

Compiling

Include the MachAPI.h file in your source files.

For the Windows Operationg System:

The dynamic C runtime should be used. /MD and /MDd for Visual Studio.

There are a few environment variables that can be used:

- **MACHSDK** should point to the Mach SDK folder. This environment variable can then be used in your projects to point to the inlude and lib directories in your project. e.g \$(MACHSDK)\include and \$(MACHSDK)\lib.
- **M4HOBBY** or should point to the Mach installation directory. Plugin projects can then use this environment variable to copy the resulting plugin the Plugins folder. e.g. In the post build event, add `copy "$(TargetPath)" "$(M4HOBBY)Plugins"`

Several compile time definitions are required or can be useful:

- Define **MACH_STATIC** in your project or define **MACH_STATIC** in your code before the MachAPI.h file is included. **This is required!**
- **_ATL_XP_TARGETING**: If you are using VS2013, it will not compile an executable that runs on windows XP by default. Defining **_ATL_XP_TARGETING** will correct this in conjuction with setting the linker minimum version to 5.01.
- **FAST_DEBUG**: A link library (MachAPIfd.lib) that uses the non debug version of the CRT (/MD) is provided for speed. Your code should then also be compiled with /MD and define FAST_DEBUG but you should turn on symbol information and link with debugging info. This keeps the debug CRT heap walking/memory checking routines from slowing things down in the debugger. Of course, it is a good idea

to check your project with full debug settings periodically to check for memory leaks and such.

- **_CRT_SECURE_NO_WARNINGS:** if you use a lot of the old C style functions that are considered as insecure, this will reduce your warning messages.

Linking

For the Windows Operationg System:

- For a Relase target, add MachAPI.lib to your library list.
- For a FastDebug target, add MachAPIfd.lib to your library list.
- For a Debug target, add MachAPId.lib to your library list.



Home



Previous



Next

API Error Return Codes

- [Return Codes by Number](#)
- [Return Codes by Definition](#)

Return Codes by Number

Error Number	Error Definition	Error Description
0	MERROR_NOERROR	No Error.
-1	MERROR_INVALID_INSTANCE	An invalid instance number or handle was specified.
-2	MERROR_INVALID_ARG	An invalid argument was specified.
-3	MERROR_INVALID_DIR	An invalid directory was specified.
-4	MERROR_INVALID_PROFILE	An invalid profile was specified.
-5	MERROR_FILE_EXCEPTION	An exception was encountered when processing the file.
-6	MERROR_FILE_EMPTY	An empty file was specified.
-7	MERROR_FILE_SHARING	A file sharing violation occurred.
-8	MERROR_FILE_INVALID	An invalid file was specified.
-9	MERROR_FILE_BADSIZE	The file specified is too large.
-10	MERROR_REGEN_DONE	The tool path regeneration is complete.
-11	MERROR_NODATA	The API function returned no data.
-12	MERROR_GRID_ACTIVE	The grid is active. (not used)
-13	MERROR_NOT_IMPLEMENTED	The function is not implemented in this version of the API.
-14	MERROR_MOTOR_NOT_FOUND	The specified motor was not found.
-2	MERROR_INVALID_PARAM	Same as MERROR_INVALID_ARG

-16	MERROR_AXIS_NOT_FOUND	The specified axis was not found.
-17	MERROR_API_INIT	The API failed to initialize properly.
-18	MERROR_NOT_NOW	The API function cannot be called at this time.
-19	MERROR_NOT_CREATED	The object was not created.
-20	MERROR_SIGNAL_NOT_FOUND	The specified signal was not found.
-21	MERROR_IO_NOT_FOUND	The specified input or output was not found.
-22	MERROR_SPIN_RANGE_NOT_FOUND	The specified spindle range was not found.
-23	MERROR_PLUGIN_NOT_FOUND	The specified plugin was not found.
-24	MERROR_DEVICE_NOT_FOUND	The specified device was not found.
-25	MERROR_INVALID_STATE	The state of the control is invalid for the API function.
-26	MERROR_AXIS_NOT_ENABLED	The specified axis is not enabled.
-27	MERROR_REG_NOT_FOUND	The specified register was not found.
-28	MERROR_IPC_NOT_READY	The Interprocess Communication subsystem is not ready.
-29	MERROR_NO_ROOM_AVALABLE	There are no more slots available for another override axis.
-30	MERROR_NOT_COMPILED	The macros did not compile correctly.
-31	MERROR_NOT_ENABLED	The control is not enabled.
-32	MERROR_SCRIPT_KILLED	The current running script has been killed.
-33	MERROR_FILE_NOT_FOUND	The specified file cannot be found.
-34	MERROR_LIC_FEATURE_NOT_FOUND	The specified license feature was not found.
-35	MERROR_LIC_REQUIREMENT_NOT_FOUND	The specified license requirement was not found.
-36	MERROR_LIC_EXPIRED	The license has expired.

-37	MERROR_LIC_BAD_ID	A bad ID was found in the license file.
-38	MERROR_LIC_BAD_KEY	A bad KEY was found in the license file.
-39	MERROR_TOOLPATH_NOT_FOUND	No tool paths exists. (informational and not indicative of an error)
-40	MERROR_TIMED_OUT	The operation timed out.
-41	MERROR_SOFTLIMITS	A machine soft limit was violated.
-42	MERROR_FILE_BADFORMAT	The format specifid is invalid.
-43	MERROR_INVALID_TYPE	The type specified is invalid.
-44	MERROR_TLM_ALLOC_FAILED	There is not enough memory available for the operation.
-45	MERROR_TLM_TOOL_UNMANAGED	The specified tool is unmanaged.
-46	MERROR_TLM_GROUP_EXPIRED	The specified tool group is expired.
-47	MERROR_TLM_GROUP_NOT_FOUND	The specified tool group was not found.
-48	MERROR_TLM_GROUP_ADDED	The specified tool group was added.
-49	MERROR_TLM_GROUP_CHANGED	The specified tool group was changed.
-50	MERROR_TLM_TOOL_NOT_FOUND	The specified tool was not found.
-51	MERROR_TLM_TOOL_ADDED	The specified tool was added.
-52	MERROR_TLM_TOOL_CHANGED	The specified tool was changed.
-53	MERROR_TLM_TOOL_EXPIRED	The specified tool has expired.
-54	MERROR_WIN_NOT_FOUND	The specified window was not found.

Return Codes by Definition

Error Definition	Error Number	Error Description
MERROR_NOERROR	0	No Error.
MERROR_INVALID_INSTANCE	-1	An invalid instance number or handle was specified.
MERROR_INVALID_ARG	-2	An invalid argument was specified.
MERROR_INVALID_DIR	-3	An invalid directory was specified.
MERROR_INVALID_PROFILE	-4	An invalid profile was specified.
MERROR_FILE_EXCEPTION	-5	An exception was encountered when processing the file.
MERROR_FILE_EMPTY	-6	An empty file was specified.
MERROR_FILE_SHARING	-7	A file sharing violation occurred.
MERROR_FILE_INVALID	-8	An invalid file was specified.
MERROR_FILE_BADSIZE	-9	The file specified is too large.
MERROR_REGEN_DONE	-10	The tool path regeneration is complete.
MERROR_NODATA	-11	The API function returned no data.
MERROR_GRID_ACTIVE	-12	The grid is active. (not used)
MERROR_NOT_IMPLEMENTED	-13	The function is not implemented in this version of the API.
MERROR_MOTOR_NOT_FOUND	-14	The specified motor was not found.
MERROR_INVALID_PARAM	-2	Same as MERROR_INVALID_ARG

MERROR_AXIS_NOT_FOUND	-16	The specified axis was not found.
MERROR_API_INIT	-17	The API failed to initialize properly.
MERROR_NOT_NOW	-18	The API function cannot be called at this time.
MERROR_NOT_CREATED	-19	The object was not created.
MERROR_SIGNAL_NOT_FOUND	-20	The specified signal was not found.
MERROR_IO_NOT_FOUND	-21	The specified input or output was not found.
MERROR_SPIN_RANGE_NOT_FOUND	-22	The specified spindle range was not found.
MERROR_PLUGIN_NOT_FOUND	-23	The specified plugin was not found.
MERROR_DEVICE_NOT_FOUND	-24	The specified device was not found.
MERROR_INVALID_STATE	-25	The state of the control is invalid for the API function.
MERROR_AXIS_NOT_ENABLED	-26	The specified axis is not enabled.
MERROR_REG_NOT_FOUND	-27	The specified register was not found.
MERROR_IPC_NOT_READY	-28	The Interprocess Communication subsystem is not ready.
MERROR_NO_ROOM_AVAILABLE	-29	There are no more slots available for another override axis.
MERROR_NOT_COMPILED	-30	The macros did not compile correctly.
MERROR_NOT_ENABLED	-31	The control is not enabled.
MERROR_SCRIPT_KILLED	-32	The current running script has been killed.
MERROR_FILE_NOT_FOUND	-33	The specified file cannot be found.
MERROR_LIC_FEATURE_NOT_FOUND	-34	The specified license feature was not found.
MERROR_LIC_REQUIREMENT_NOT_FOUND	-35	The specified license requirement was not found.
MERROR_LIC_EXPIRED	-36	The license has expired.

MERROR_LIC_BAD_ID	-37	A bad ID was found in the license file.
MERROR_LIC_BAD_KEY	-38	A bad KEY was found in the license file.
MERROR_TOOLPATH_NOT_FOUND	-39	No tool paths exists. (informational and not indicative of an error)
MERROR_TIMED_OUT	-40	The operation timed out.
MERROR_SOFTLIMITS	-41	A machine soft limit was violated.
MERROR_FILE_BADFORMAT	-42	The format specifid is invalid.
MERROR_INVALID_TYPE	-43	The type specified is invalid.
MERROR_TLM_ALLOC_FAILED	-44	There is not enough memory available for the operation.
MERROR_TLM_TOOL_UNMANAGED	-45	The specified tool is unmanaged.
MERROR_TLM_GROUP_EXPIRED	-46	The specified tool group is expired.
MERROR_TLM_GROUP_NOT_FOUND	-47	The specified tool group was not found.
MERROR_TLM_GROUP_ADDED	-48	The specified tool group was added.
MERROR_TLM_GROUP_CHANGED	-49	The specified tool group was changed.
MERROR_TLM_TOOL_NOT_FOUND	-50	The specified tool was not found.
MERROR_TLM_TOOL_ADDED	-51	The specified tool was added.
MERROR_TLM_TOOL_CHANGED	-52	The specified tool was changed.
MERROR_TLM_TOOL_EXPIRED	-53	The specified tool has expired.
MERROR_WIN_NOT_FOUND	-54	The specified window was not found.



Home



Previous



Next

API Reference by Category

The API documentation has been devided up into categories to help quickly locate API information for a given programming task.

- [Plugins and Devices](#): API functions that are primarily used to create plugins with devices.
- [Mach I/O](#): API functions that are primarily used to interface with I/O and to create I/O plugins.
- [Mach Signals](#): API functions that work with Mach Signals.
- [Mach Registers](#): API functions that work with Mach Registers.
- [Motors](#): API functions that work with Motors.
- [Screw Mapping](#): API functions that work with Motor screw maps.
- [Axes](#): API functions that work with Axes.
- [Motion](#): API functions that are primarily used to create motion plugins.
- [Operation](#): API functions that are used to operate the machine (Cycle start, Feed Hold, etc...).
- [GUI](#): API functions that are specific to GUI programming.
- [Profile \(INI\) Settings](#): API functions for profile INI manipulation.
- [General](#): General API functions.

- [Gcode File](#): API functions to manipulate Gcode files.
- [Tool Offsets](#): API functions dealing with tool offsets and tool selection.
- [Jogging](#): Jogging API functions.
- [MPGs](#): API functions for working with MPGs.
- [Soft Limits](#): API functions for working with Soft Limits.
- [Spindle](#): Spindle API functions.
- [Scripting](#): API functions used for manipulating scripts.
- [Status Messages](#): API functions for reporting error and status messages.
- [License](#): API functions for working with license files.

Plugins and Devices

The Mach core uses loadable modules known as "plugins" that provide interfaces to devices and/or provides services. When the core is initialized, the Plugins directory is searched and each plugin found is loaded with the following sequence:

1. The `mcPluginLoad()` entry point is called. The plugin should register itself with the core and provide its capabilities at this time.
2. The `mcPluginInit()` entry point is called for each Core instance. Devices should be registered at this time as well as any I/O and registers that belong to the device.

- [mcDeviceGetHandle](#)
- [mcDeviceGetInfo](#)
- [mcDeviceGetInfoStruct](#)
- [mcDeviceGetNextHandle](#)
- [mcDeviceRegister](#)
- [mcPluginConfigure](#)
- [mcPluginCoreLoad](#)
- [mcPluginCoreUnload](#)

- [mcPluginDiagnostic](#)
- [mcPluginEnable](#)
- [mcPluginGetEnabled](#)
- [mcPluginGetInfoStruct](#)
- [mcPluginGetLicenseFeature](#)
- [mcPluginGetNextHandle](#)
- [mcPluginGetValid](#)
- [mcPluginInstall](#)
- [mcPluginRegister](#)
- [mcPluginRemove](#)

```
int mcDeviceGetHandle(  
    MINSTANCE mInst, int devid, HMCDEV *hDev);  
  
hDev, rc = mc.mcDeviceGetHandle(  
    number mInst, number devid)  
  
int mInst = 0;  
  
int devid = 0;  
  
HMCDEV hDev;  
  
//See if we can find a device with an I of zero int rc = <font  
color="blue">mcDeviceGetHandle(mInst, devid, &hDev);</font> if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// We found it!</font> }  
 
```

```

mcDeviceGetInfo(
    HMCDEV hDev, char *nameBuf, size_t nameBuflen, char *descBuf,
    size_t descBuflen, int *type, int *id);

nameBuf, descBuf, type, id, rc = mc.mcDeviceGetInfo(HMCDEV hDev}

int mInst=0;

HMCSIG hDev = 0;

devinfo_t devinf;

char nameBuf[80];

char descBuf[80];

int type = 0;

int id = 0;

//Look for all the IO devices and get their devinfo_t struct while
(mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev) ==
MERROR_NOERROR) {

    if (hSig != 0) {

        <font color="blue">mcDeviceGetInfo(hDev, nameBuf, sizeof(nameBuf),
descBuf, sizeof(descBuf), &type, &id);</font> <font color="green">// do
something with the device info.</font> } else {

        break; }

}

```

```
int mcDeviceGetInfoStruct(  
    HMCSIG hDev, devinfo_t *devinf);  
  
N/A  
  
struct devinfo {  
    char devName[80]; char devDesc[80]; int devType; int devId; };  
  
typedef struct devinfo devinfo_t;  
  
int mInst=0;  
  
HMCSIG hDev = 0; devinfo_t devinf;  
  
//Look for all the IO devices and get their devinfo_t struct while  
(mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcDeviceGetInfoStruct(hDev, devinf); </font> <font  
        color="green">// Do something with the device info.</font> } else {  
  
        break; }  
  
    }  
}
```

```
int mcDeviceGetNextHandle(
    MINSTANCE mInst, int devtype, HMCDEV startDev, HMCDEV
*hDev);

hDev, rc = mc.mcDeviceGetNextHandle(
    number mInst, number devtype, number startDev)

int mInst=0;

HMCDEV hDev;

<font color="green">/>Look for all the IO devices</font> while (<font
color="blue">mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev,
&hDev)</font> == MERROR_NOERROR) {

    if (hSig != 0) {

        <font color="green">/> Do something with hDev.</font> } else {

            break; }

}
```

mcDeviceRegister(

 MINSTANCE mInst, HMCPLUG plugin, const char *DeviceName,
 const char *DeviceDesc, int Type, HMCDEV *hDev);

N/A

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {

 m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime
 = .001;

 mcDeviceRegister(m_cid, m_id, "Sim0",
 "Simulation Device", DEV_TYPE_MOTION | DEV_TYPE_IO,
 &m_hDev);

 mcRegisterIo(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,
 &m_Input0); mcRegisterIo(m_hDev, "Output0", "Output0",
 IO_TYPE_OUTPUT, &m_Output0);

 if (!m_timer->Start(10, wxTIMER_ONE_SHOT)) {

 wxMessageBox(wxT("Timer could not start!"), wxT("Timer")); }

}

```
int mcPluginConfigure(  
    MINSTANCE mInst, int plugId);
```

N/A

```
MINSTANCE mInst = 0;  
  
HMCPLUG hPlug = 0;  
  
plugininfo_t plugininf;
```

```
<font color="green">// Loop through all of the plugins looking for  
plugins<br/>// that have configuration dialogs.</font> while  
(mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug) ==  
MERROR_NOERROR) {  
  
    <font color="green">// Get the plugin info that contains the plugin ID.  
</font> rc = mcPluginGetInfoStruct(hPlug, &plugininf); if (rc ==  
MERROR_NOERROR) {  
  
    int pluginId = plugininf.plugId; <font color="green">// Call the  
configuration dialog.</font> rc = <font  
color="blue">mcPluginConfigure(mInst, pluginId);</font>; }  
  
}
```

```
int mcPluginCoreLoad(
```

```
    const char *shortName);
```

N/A

```
<font color="green">// Load the mcGalil plugin.</font> int rc =  
mcPluginCoreLoad("mcGalil");
```

```
int mcPluginCoreUnload(
```

```
    const char *shortName);
```

N/A

```
<font color="green">// Unload the mcGalil plugin.</font> int rc =  
mcPluginCoreUnload("mcGalil");
```

```
int mcPluginDiagnostic(  
    MINSTANCE mInst, int pluginId);
```

N/A

```
MINSTANCE mInst = 0;
```

```
HMCPLUG hPlug = 0;
```

```
plugininfo_t plugininf;
```

```
<font color="green">// Loop through all of the plugins looking for  
plugins<br/>// that have diagnostic dialogs.</font> while  
(mcPluginGetNextHandle(PLUG_TYPE_DIAG, hPlug, &hPlug) ==  
MERROR_NOERROR) {
```

```
    <font color="green">// Get the plugin info that contains the plugin ID.  
</font> rc = mcPluginGetInfoStruct(hPlug, &plugininf); if (rc ==  
MERROR_NOERROR) {
```

```
        int pluginId = plugininf.pluginId; <font color="green">// Call the diagnostics  
dialog.</font> rc = <font color="blue">mcPluginDiagnostic(mInst,  
pluginId);</font>; }
```

```
}
```

```
int mcPluginEnable(  
    HMCPLUG hPlug, BOOL enable);
```

N/A

```
HMCPLUG hPlug = 0; // Loop through all of the  
plugins looking for plugins<br/>// that have configuration dialogs.</font>  
while (mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug)  
== MERROR_NOERROR) {  
  
    // Enable the plugin.</font> rc = <font  
    color="blue">mcPluginEnable(hPlug, TRUE)</font>; }
```

```
int mcPluginGetEnabled(
```

```
    HMCPLUG hPlug, BOOL *enabled);
```

N/A

```
HMCPLUG hPlug = 0;
```

```
BOOL enabled = FALSE;
```

```
<font color="green">// Loop through all of the plugins looking for  
plugins<br/>// that have configuration dialogs.</font> while  
(mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug) ==  
MERROR_NOERROR) {
```

```
    <font color="green">// See if the plugin is enabled..</font> rc = <font  
color="blue">mcPluginGetEnabled(hPlug, &enabled)</font>; if (rc ==  
MERROR_NOERROR) {
```

```
        if (enabled == TRUE) {
```

```
            <font color="green">// The plugin is enabled!!!</font> } else {
```

```
            <font color="green">// The plugin is not enabled!!!</font> }
```

```
}
```

```
}
```

```
mcPluginGetInfoStruct(HMCPLUG hPlug, pluginfo_t *pluginf);
```

N/A

```
HMCPLUG hPlug = 0; pluginfo_t pluginf;
```

```
<font color="green">// Loop through all of the plugins.
```

```
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug) ==  
MERROR_NOERROR) {
```

```
    <font color="green">// Get the plugin info that contains the plugin ID.  
</font> rc = <font color="blue">mcPluginGetInfoStruct(hPlug, &pluginf)  
</font>; if (rc == MERROR_NOERROR) {
```

```
        int pluginId = pluginf.pluginId; ...
```

```
}
```

```
}
```

```
</font>
```

```
int mcPluginGetLicenseFeature(
```

```
    HMCPLUG hPlug, int index, char *buf, int bufSize, BOOL *status);
```

N/A

```
HMCPLUG hPlug = 0;
```

```
<font color="green">// Loop through all of the plugins.
```

```
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug) ==  
MERROR_NOERROR) {
```

```
    <font color="green">// Get all of the plugin license features</font> int  
    index = 0; char buf[80]; BOOL licensed = FALSE; while (<font  
    color="blue">mcPluginGetLicenseFeature(hPlug, index, buf, sizeof(buf),  
    &licensed)</font> == MERROR_NOERROR) {
```

```
        printf("license feature %s is %s.\n", buf, licensed == TRUE ? "licensed" :  
        "not licensed"); index++; }
```

```
}
```

```
</font>
```

```
int mcPluginGetNextHandle(
```

```
    int plugType, HMCPLUG startPlug, HMCPLUG *hPlug);
```

N/A

```
HMCPLUG hPlug = 0;
```

```
<font color="green">// Loop through all of the plugins looking for  
plugins<br/>// that have configuration dialogs and provide I/O.</font> int  
attr = PLUG_TYPE_CONFIG | PLUG_TYPE_IO; while (<font  
color="blue">mcPluginGetNextHandle(attr, hPlug, &hPlug)</font> ==  
MERROR_NOERROR) {
```

```
    <font color="green">// Enable the plugin.</font> rc =  
    mcPluginEnable(hPlug, TRUE); }
```

```
int mcPluginGetValid(
```

```
    HMCPLUG hPlug, BOOL *valid);
```

N/A

```
HMCPLUG hPlug = 0;
```

```
BOOL valid = FALSE;
```

```
<font color="green">// Loop through all of the plugins checking to see if  
they are valid.</font> while (mcPluginGetNextHandle(PLUG_TYPE_ALL,  
hPlug, &hPlug) == MERROR_NOERROR) {
```

```
    <font color="green">// Check the plugin validity.</font> rc = <font  
color="blue">mcPluginGetValid(hPlug, &valid)</font>; if (rc ==  
MERROR_NOERROR) {
```

```
        if (valid == TRUE) {
```

```
            <font color="green">// The plugin is valid and will function.</font> }  
        else {
```

```
            <font color="green">// The plugin is invalid and will not function.  
</font> }
```

```
    }
```

```
}
```

```
int mcPluginInstall(  
    const char *m4plug);
```

N/A

```
<font color="green">// Install the Windows mcGalil plugin from an  
installation archive.</font> int rc = mcPluginInstall("mcGalil.m4Plugw");  
if (rc == MERROR_NOERROR) {  
  
<font color="green">//The plugin installed successfully.</font> }
```

```
int mcPluginRegister(
```

```
    HMCPLUG hPlug, const char *DeveloperId, const char *Desc, const
    char *Version, int Type);
```

N/A

```
<font color="green">// This function gets called (only once) when the
Plugin is loaded by Mach Core.</font> MCP_API int MCP_APIENTRY
mcPluginLoad(HMCPLUG id) {
```

...

```
<font color="blue">mcPluginRegister(id, "Newfangled", "Simulator -
Newfangled Solutions", MC_VERSION_STR, PLUG_TYPE_MOTION |
PLUG_TYPE_IO | PLUG_TYPE_REG | PLUG_TYPE_CONFIG |
PLUG_TYPE_DIAG)</font>; return(MERROR_NOERROR); }
```

```
int mcPluginRemove(const char *shortName);
```

N/A

```
<font color="green">// Remove the mcGalil plugin.</font> MINSTANCE  
mInst = 0; int rc = mcPluginRemove("mcGalil");
```

Mach I/O

I/O is registered by devices that are registered by plugins. The Mach core signals can then be mapped to the I/O. The `mcIoGetHandle()` function can be used to get a handle to an I/O object. It takes a path in the form of "**device/ioName**". Once a handle is obtained, the state can be checked or set depending if the I/O object is an input or an output. Scripts can operate with I/O directly or through the signal interface, whichever is the most useful for a given task.

- [mcIoGetHandle](#)
- [mcIoGetInfoStruct](#)
- [mcIoGetNextHandle](#)
- [mcIoGetState](#)
- [mcIoGetType](#)
- [mcIoGetUserData](#)
- [mcIoIsEnabled](#)
- [mcIoRegister](#)
- [mcIoSetDesc](#)
- [mcIoSetName](#)
- [mcIoSetState](#)

- [mcIoSetType](#)
- [mcIoSetUserData](#)
- [mcIoSyncSignal](#)
- [mcIoUnregister](#)

```
mcIoGetHandle(  
    MINSTANCE mInst, const char *path, HMCIO *hIo);  
  
hIo, rc = mc.mcIoGetHandle(  
    number mInst, string path)  
  
HMCIO hIo;  
  
char *path = "Sim0/Input0";  
  
<font color="blue">mcIoGetHandle(m_cid, path, &hIo);</font>
```

```

int mcIoGetInfoStruct(
    HMCIO hIo, ioinfo_t *ioinf);

ioinf, rc = mc.mcIoGetInfoStruct(
    number hIo)

struct ioinfo {
    char ioName[80]; char ioDesc[80]; int ioType; HMCDEV ioDev;
    HMCSIG ioMappedSignals[MAX_MAPPED_SIGNAL]; void
    *ioUserData; int ioInput; };

typedef struct ioinfo ioinfo_t;

HMCDEV hDev = 0;

devinfo_t devinf;

int rc;

<font color="green">// Get all IO information from every registered device.
</font> while (mcDeviceGetNextHandle(0, DEV_TYPE_IO, hDev,
&hDev) == MERROR_NOERROR) {

    if (mcDeviceGetInfoStruct(hDev, &devinf) == MERROR_NOERROR) {

        HMCIO hIo = 0; while (mcIoGetNextHandle(hDev, hIo, &hIo) ==
MERROR_NOERROR) {

            ioinfo_t ioinf; rc = <font color="blue">mcIoGetInfoStruct(hIo, &ioinf);
</font> if (rc ==

```

```
MERROR_NOERROR No Error.) {  
    <font color="green">// IO information successfully retrieved.</font> }  
}  
}  
}
```

```
mcIoGetNextHandle(  
    HMCDEV hDev, HMCIO startIo) HMCIO *hIo)  
  
HMCDEV hDev = 0;  
  
devinfo_t devinf;  
  
int rc;  
  
  
  
<font color="green">// Get all IO information from every registered device.  
</font> while (mcDeviceGetNextHandle(0, DEV_TYPE_IO, hDev,  
&hDev) ==  
  
    MERROR_NOERROR No Error.) {  
  
    if (mcDeviceGetInfoStruct(hDev, &devinf) ==  
  
        MERROR_NOERROR No Error.) {  
  
        HMCIO hIo = 0; while (<font color="blue">mcIoGetNextHandle(hDev,  
        hIo, &hIo)</font> ==  
  
            MERROR_NOERROR No Error.) {  
  
            ioinfo_t ioinf; rc = mcIoGetInfoStruct(hIo, &ioinf); if (rc ==  
  
                MERROR_NOERROR No Error.) {
```

```
<font color="green">// IO information successfully retrieved.</font> }  
}  
}  
}
```

```
mcIoGetState(  
    HMCIO hIo,  
    bool *state);  
  
simControl::simControl(MINSTANCE mInst, HMCPLUG id) {  
  
    m_cid = mInst; m_id = id;  
  
    m_timer = new simTimer(this); m_cycletime = .001; bool oldstate =  
    false; bool newstate = false;  
  
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",  
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);  
  
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,  
    &m_Input0); <font color="blue">mcIoGetState(m_Input0, &oldstate);  
    </font> newstate = simGetIO(0); fi (newstate != oldstate) {  
  
        mcIoSetState(m_Input0, newstate); }  
}
```

```
mcIoGetType(  
    HMCIO hIo, int *type);  
  
type, rc = mcIoGetType(  
    number hIo)  
  
simControl::simControl(MINSTANCE mInst, HMCPLUG id) {  
  
    m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime  
    = .001; bool oldstate = false; bool newstate = false;  
  
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",  
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);  
  
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,  
    &m_Input0); <font color="blue">mcIoGetType(m_Input0, &type);</font>  
    <font color="green">// type will equal IO_TYPE_INPUT</font> }
```

```
mcIoGetUserData(  
    HMCIO hIo, void **data);
```

N/A

```
struct myData {  
    int myPortNumber; int myPinNumber; };
```

```
void GetUserData(void) {  
  
    MINSTANCE mInst = 0; HMCIO hIo; void *data; if  
(mcIoGetHandle(mInst, "Sim0/Input0", &hIo) == MERROR_NOERROR)  
{  
  
    <font color="blue">mcIoGetUserData(hIo, &data);</font> <font  
color="green">// type cast the pointer</font> struct myData *mData =  
(struct myData *)data; }  
  
}
```

```
mcIoIsEnabled (  
    HMCIO hIo, bool *enabled);  
  
HMCIO hIo;  
  
char *path = "Sim0/Input0"; bool enabled; mcIoGetHandle(m_cid, path,  
&hIo);  
//Get the handle  
mcIoIsEnabled(hIo ,  
&enabled);  
//Get the enabled state of the IO
```

mcIoRegister(

 HMCDEV hDev, const char *IoName, const char *IoDesc, int Type,
 HMCIO *hIo);

N/A

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {

 m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime
 = .001;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",
 DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);

 mcIoRegister(m_hDev, "Input0", "Input0",
 IO_TYPE_INPUT, &m_Input0); mcIoRegister(m_hDev, "Output0",
 "Output0", IO_TYPE_OUTPUT, &m_Output0);

 if (!m_timer->Start(10, wxTIMER_ONE_SHOT)) {

 wxMessageBox(wxT("Timer could not start!"), wxT("Timer")); }

}

```
mcIoSetDesc(  
    HMCIO hIo, const char *desc);  
  
rc = mc.mcIoSetDesc(  
    number hIo, string desc)  
  
HMCIO hIo;  
  
char *path = "Sim0/Input0";  
  
if (mcIoGetHandle(m_cid, path, &hIo) == MERROR_NOERROR) {  
    <font color="blue">mcIoSetDesc(hIo, "my mew description);</font> }  
}
```

```
mcIoSetName(  
    HMCIO hIo, const char *name);  
  
rc = mc.mcIoSetName(  
    number hIo, string name)  
  
HMCIO hIo;  
  
char *path = "Sim0/Input0";  
  
if (mcIoGetHandle(m_cid, path, &hIo) == MERROR_NOERROR) {  
  
    <font color="blue">mcIoSetName(hIo, "Limit0++");</font> <font  
    color="green">// The name is changed from Input0 to Limit0++</font> }  
}
```

```
mcIoSetState(  
    HMCIO hIo,  
    bool state);  
  
simControl::simControl(MINSTANCE mInst, HMCPLUG id) {  
  
    m_cid = mInst; m_id = id;  
  
    m_timer = new simTimer(this); m_cycletime = .001; bool oldstate =  
    false; bool newstate = false;  
  
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",  
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);  
  
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,  
    &m_Input0); <font color="blue">mcIoGetState(m_Input0, &oldstate);  
    </font> newstate = simGetIO(0); if (newstate != oldstate) {  
  
        mcIoSetState(m_Input0, newstate); }  
    }  
}
```

```
mcIoSetType(  
    HMCIO hIo,  
    int type);  
  
rc = mcIoSetType(  
    number hIo  
    number type)  
  
simControl::simControl(MINSTANCE mInst, HMCPLUG id) {  
    m_cid = mInst; m_id = id;  
  
    m_timer = new simTimer(this); m_cycletime = .001; bool oldstate =  
    false; bool newstate = false;  
  
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",  
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);  
  
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,  
    &m_Input0); <font color="blue">mcIoSetType(m_Input0,  
    IO_TYPE_OUTPUT);</font> <font color="green">// change the type to  
    IO_TYPE_OUTPUT</font> }
```

```
mcIoSetUserData(  
    HMCIO hIo, void **data);
```

N/A

```
struct myData {  
    int myPortNumber; int myPinNumber; };
```

```
struct myData data;
```

```
void GetUserData(void) {  
  
    MINSTANCE mInst = 0; HMCIO hIo; struct myData data;  
    data.myPortNumber = 1; data.myPinNumber = 12; if  
(mcIoGetHandle(mInst, "Sim0/Input0", &hIo) == MERROR_NOERROR)  
{  
  
    <font color="blue">mcIoSetUserData(hIo, &data);</font> }  
  
}
```

mcIoSyncSignal(

 HMCIO hIo,

 BOOL state);

N/A

N/A

```
mcIoUnregister(  
    HMCDEV hDev, HMCIO hIo);
```

N/A

N/A

Mach Signals

The Mach core communicates with the outside world via a signal mechanism. All signals are "owned" by the core. The core drives output signals and input signals direct the core. This means that a user should never set an output signal and the core will never set an input signal.

It is important to note that signals are not I/O. However, output signals can drive outputs and inputs can drive input signals if they are mapped to repetitive I/O objects. This mapping is performed with `mcSignalMap()`. The I/O device has no need to know what signals to which any I/O may be mapped. This allows for abstraction of the signal functions.

Signals operations use a handle that represents the signal object in the core. This handle can be obtained via `mcSignalGetHandle()` by providing a signal ID. Signal IDs are defined in the `MachAPI.h` header file. Input signal ID definitions begin with **ISIG** and output signal ID definitions begin with **OSIG**.

The state of input and output signals can be retrieved via `mcSignalGetState()`. The state of input signals can be set via `mcSignalSetState()`.

Signal information can be retrieved via `mcSignalGetInfo()` and `mcSignalGetInfoStruct()`.

- [Input Signals](#)
- [Output Signals](#)
- [mcSignalEnable](#)

- [mcSignalGetHandle](#)
- [mcSignalGetInfo](#)
- [mcSignalGetInfoStruct](#)
- [mcSignalGetNextHandle](#)
- [mcSignalGetState](#)
- [mcSignalHandleWait](#)
- [mcSignalMap](#)
- [mcSignalSetActiveLow](#)
- [mcSignalSetState](#)
- [mcSignalUnmap](#)
- [mcSignalWait](#)

Input Signals

Signal	Description
ISIG_PROBE	Probe
ISIG_PROBE1	Probe1
ISIG_PROBE2	Probe2
ISIG_PROBE3	Probe3
ISIG_INDEX	Index
ISIG_INPUT0	Input #0
ISIG_INPUT1	Input #1
ISIG_INPUT2	Input #2
ISIG_INPUT3	Input #3
ISIG_INPUT4	Input #4
ISIG_INPUT5	Input #5
ISIG_INPUT6	Input #6
ISIG_INPUT7	Input #7
ISIG_INPUT8	Input #8
ISIG_INPUT9	Input #9
ISIG_INPUT10	Input #10
ISIG_INPUT11	Input #11
ISIG_INPUT12	Input #12
ISIG_INPUT13	Input #13
ISIG_INPUT14	Input #14
ISIG_INPUT15	Input #15
ISIG_INPUT16	Input #16
ISIG_INPUT17	Input #17

ISIG_INPUT18	Input #18
ISIG_INPUT19	Input #19
ISIG_INPUT20	Input #20
ISIG_INPUT21	Input #21
ISIG_INPUT22	Input #22
ISIG_INPUT23	Input #23
ISIG_INPUT24	Input #24
ISIG_INPUT25	Input #25
ISIG_INPUT26	Input #26
ISIG_INPUT27	Input #27
ISIG_INPUT28	Input #28
ISIG_INPUT29	Input #29
ISIG_INPUT30	Input #30
ISIG_INPUT31	Input #31
ISIG_INPUT32	Input #32
ISIG_INPUT33	Input #33
ISIG_INPUT34	Input #34
ISIG_INPUT35	Input #35
ISIG_INPUT36	Input #36
ISIG_INPUT37	Input #37
ISIG_INPUT38	Input #38
ISIG_INPUT39	Input #39
ISIG_INPUT40	Input #40
ISIG_INPUT41	Input #41
ISIG_INPUT42	Input #42
ISIG_INPUT43	Input #43
ISIG_INPUT44	Input #44
ISIG_INPUT45	Input #45
ISIG_INPUT46	Input #46

ISIG_INPUT47	Input #47
ISIG_INPUT48	Input #48
ISIG_INPUT49	Input #49
ISIG_INPUT50	Input #50
ISIG_INPUT51	Input #51
ISIG_INPUT52	Input #52
ISIG_INPUT53	Input #53
ISIG_INPUT54	Input #54
ISIG_INPUT55	Input #55
ISIG_INPUT56	Input #56
ISIG_INPUT57	Input #57
ISIG_INPUT58	Input #58
ISIG_INPUT59	Input #59
ISIG_INPUT60	Input #60
ISIG_INPUT61	Input #61
ISIG_INPUT62	Input #62
ISIG_INPUT63	Input #63
ISIG_LIMITOVER	Limit Override
ISIG_EMERGENCY	E-Stop
ISIG_THCON	THC On
ISIG_THCUP	THC Up
ISIG_THCDOWN	THC Down
ISIG_TIMING	Timing
ISIG_JOGXp	Jog X+
ISIG_JOGXN	Jog X-
ISIG_JOGYP	Jog Y+
ISIG_JOGYN	Jog Y-
ISIG_JOGZP	Jog Z+

ISIG_JOGZN	Jog Z-
ISIG_JOGAP	Jog A+
ISIG_JOGAN	Jog A-
ISIG_JOGBP	Jog B+
ISIG_JOGBN	Jog B-
ISIG_JOGCP	Jog C+
ISIG_JOGCN	Jog C-
ISIG_SPINDLE_AT_SPEED	Spindle At Speed
ISIG_SPINDLE_AT_ZERO	Spindle At Zero
ISIG_MOTOR0_HOME	Motor 0 Home
ISIG_MOTOR0_PLUS	Motor 0 ++
ISIG_MOTOR0_MINUS	Motor 0 --
ISIG_MOTOR1_HOME	Motor 1 Home
ISIG_MOTOR1_PLUS	Motor 1 ++
ISIG_MOTOR1_MINUS	Motor 1 --
ISIG_MOTOR2_HOME	Motor 2 Home
ISIG_MOTOR2_PLUS	Motor 2 ++
ISIG_MOTOR2_MINUS	Motor 2 --
ISIG_MOTOR3_HOME	Motor 3 Home
ISIG_MOTOR3_PLUS	Motor 3 ++
ISIG_MOTOR3_MINUS	Motor 3 --
ISIG_MOTOR4_HOME	Motor 4 Home
ISIG_MOTOR4_PLUS	Motor 4 ++
ISIG_MOTOR4_MINUS	Motor 4 --
ISIG_MOTOR5_HOME	Motor 5 Home
ISIG_MOTOR5_PLUS	Motor 5 ++
ISIG_MOTOR5_MINUS	Motor 5 --
ISIG_MOTOR6_HOME	Motor 6 Home
ISIG_MOTOR6_PLUS	Motor 6 ++

ISIG_MOTOR6_MINUS	Motor 6 --
ISIG_MOTOR7_HOME	Motor 7 Home
ISIG_MOTOR7_PLUS	Motor 7 ++
ISIG_MOTOR7_MINUS	Motor 7 --
ISIG_MOTOR8_HOME	Motor 8 Home
ISIG_MOTOR8_PLUS	Motor 8 ++
ISIG_MOTOR8_MINUS	Motor 8 --
ISIG_MOTOR9_HOME	Motor 9 Home
ISIG_MOTOR9_PLUS	Motor 9 ++
ISIG_MOTOR9_MINUS	Motor 9 --
ISIG_MOTOR10_HOME	Motor 10 Home
ISIG_MOTOR10_PLUS	Motor 10 ++
ISIG_MOTOR10_MINUS	Motor 10 --
ISIG_MOTOR11_HOME	Motor 11 Home
ISIG_MOTOR11_PLUS	Motor 11 ++
ISIG_MOTOR11_MINUS	Motor 11 --
ISIG_MOTOR12_HOME	Motor 12 Home
ISIG_MOTOR12_PLUS	Motor 12 ++
ISIG_MOTOR12_MINUS	Motor 12 --
ISIG_MOTOR13_HOME	Motor 13 Home
ISIG_MOTOR13_PLUS	Motor 13 ++
ISIG_MOTOR13_MINUS	Motor 13 --
ISIG_MOTOR14_HOME	Motor 14 Home
ISIG_MOTOR14_PLUS	Motor 14 ++
ISIG_MOTOR14_MINUS	Motor 14 --
ISIG_MOTOR15_HOME	Motor 15 Home
ISIG_MOTOR15_PLUS	Motor 15 ++
ISIG_MOTOR15_MINUS	Motor 15 --

ISIG_MOTOR16_HOME	Motor 16 Home
ISIG_MOTOR16_PLUS	Motor 16 ++
ISIG_MOTOR16_MINUS	Motor 16 --
ISIG_MOTOR17_HOME	Motor 17 Home
ISIG_MOTOR17_PLUS	Motor 17 ++
ISIG_MOTOR17_MINUS	Motor 17 --
ISIG_MOTOR18_HOME	Motor 18 Home
ISIG_MOTOR18_PLUS	Motor 18 ++
ISIG_MOTOR18_MINUS	Motor 18 --
ISIG_MOTOR19_HOME	Motor 19 Home
ISIG_MOTOR19_PLUS	Motor 19 ++
ISIG_MOTOR19_MINUS	Motor 19 --
ISIG_MOTOR20_HOME	Motor 20 Home
ISIG_MOTOR20_PLUS	Motor 20 ++
ISIG_MOTOR20_MINUS	Motor 20 --
ISIG_MOTOR21_HOME	Motor 21 Home
ISIG_MOTOR21_PLUS	Motor 21 ++
ISIG_MOTOR21_MINUS	Motor 21 --
ISIG_MOTOR22_HOME	Motor 22 Home
ISIG_MOTOR22_PLUS	Motor 22 ++
ISIG_MOTOR22_MINUS	Motor 22 --
ISIG_MOTOR23_HOME	Motor 23 Home
ISIG_MOTOR23_PLUS	Motor 23 ++
ISIG_MOTOR23_MINUS	Motor 23 --
ISIG_MOTOR24_HOME	Motor 24 Home
ISIG_MOTOR24_PLUS	Motor 24 ++
ISIG_MOTOR24_MINUS	Motor 24 --
ISIG_MOTOR25_HOME	Motor 25 Home
ISIG_MOTOR25_PLUS	Motor 25 ++

ISIG_MOTOR25_MINUS	Motor 25 --
ISIG_MOTOR26_HOME	Motor 26 Home
ISIG_MOTOR26_PLUS	Motor 26 ++
ISIG_MOTOR26_MINUS	Motor 26 --
ISIG_MOTOR27_HOME	Motor 27 Home
ISIG_MOTOR27_PLUS	Motor 27 ++
ISIG_MOTOR27_MINUS	Motor 27 --
ISIG_MOTOR28_HOME	Motor 28 Home
ISIG_MOTOR28_PLUS	Motor 28 ++
ISIG_MOTOR28_MINUS	Motor 28 --
ISIG_MOTOR29_HOME	Motor 29 Home
ISIG_MOTOR29_PLUS	Motor 29 ++
ISIG_MOTOR29_MINUS	Motor 29 --
ISIG_MOTOR30_HOME	Motor 30 Home
ISIG_MOTOR30_PLUS	Motor 30 ++
ISIG_MOTOR30_MINUS	Motor 30 --
ISIG_MOTOR31_HOME	Motor 31 Home
ISIG_MOTOR31_PLUS	Motor 31 ++
ISIG_MOTOR31_MINUS	Motor 31 --

Output Signals

Signal	Description
OSIG_DIGTRIGGER	Digitize Trigger
OSIG_SPINDLEON	Spindle On
OSIG_SPINDLEFWD	Spindle Fwd
OSIG_SPINDLEREV	Spindle Rev
OSIG_COOLANTON	Coolant On
OSIG_MISTON	Mist On
OSIG_CHARGE	Charge Pump #1
OSIG_CHARGE2	Charge Pump #2
OSIG_CURRENTHILOW	Current Hi/Low
OSIG_ENABLE0	Enable #0
OSIG_ENABLE1	Enable #1
OSIG_ENABLE2	Enable #2
OSIG_ENABLE3	Enable #3
OSIG_ENABLE4	Enable #4
OSIG_ENABLE5	Enable #5
OSIG_ENABLE6	Enable #6
OSIG_ENABLE7	Enable #7
OSIG_ENABLE8	Enable #8
OSIG_ENABLE9	Enable #9
OSIG_ENABLE10	Enable #10
OSIG_ENABLE11	Enable #11
OSIG_ENABLE12	Enable #12
OSIG_ENABLE13	Enable #13

OSIG_ENABLE14	Enable #14
OSIG_ENABLE15	Enable #15
OSIG_ENABLE16	Enable #16
OSIG_ENABLE17	Enable #17
OSIG_ENABLE18	Enable #18
OSIG_ENABLE19	Enable #19
OSIG_ENABLE20	Enable #20
OSIG_ENABLE21	Enable #21
OSIG_ENABLE22	Enable #22
OSIG_ENABLE23	Enable #23
OSIG_ENABLE24	Enable #24
OSIG_ENABLE25	Enable #25
OSIG_ENABLE26	Enable #26
OSIG_ENABLE27	Enable #27
OSIG_ENABLE28	Enable #28
OSIG_ENABLE29	Enable #29
OSIG_ENABLE30	Enable #30
OSIG_ENABLE31	Enable #31
OSIG_XLIMITPLUS	X ++
OSIG_XLIMITMINUS	X --
OSIG_XHOME	X Home
OSIG_YLIMITPLUS	Y ++
OSIG_YLIMITMINUS	Y --
OSIG_YHOME	Y Home
OSIG_ZLIMITPLUS	Z ++
OSIG_ZLIMITMINUS	Z --
OSIG_ZHOME	Z Home
OSIG_ALIMITPLUS	A ++
OSIG_ALIMITMINUS	A --

OSIG_AHOME	A Home
OSIG_BLIMITPLUS	B ++
OSIG_BLIMITMINUS	B --
OSIG_BHOME	B Home
OSIG_CLIMITPLUS	C ++
OSIG_CLIMITMINUS	C --
OSIG_CHOME	C Home
OSIG_OUTPUT0	Output #0
OSIG_OUTPUT1	Output #1
OSIG_OUTPUT2	Output #2
OSIG_OUTPUT3	Output #3
OSIG_OUTPUT4	Output #4
OSIG_OUTPUT5	Output #5
OSIG_OUTPUT6	Output #6
OSIG_OUTPUT7	Output #7
OSIG_OUTPUT8	Output #8
OSIG_OUTPUT9	Output #9
OSIG_OUTPUT10	Output #10
OSIG_OUTPUT11	Output #11
OSIG_OUTPUT12	Output #12
OSIG_OUTPUT13	Output #13
OSIG_OUTPUT14	Output #14
OSIG_OUTPUT15	Output #15
OSIG_OUTPUT16	Output #16
OSIG_OUTPUT17	Output #17
OSIG_OUTPUT18	Output #18
OSIG_OUTPUT19	Output #19
OSIG_OUTPUT20	Output #20

OSIG_OUTPUT21	Output #21
OSIG_OUTPUT22	Output #22
OSIG_OUTPUT23	Output #23
OSIG_OUTPUT24	Output #24
OSIG_OUTPUT25	Output #25
OSIG_OUTPUT26	Output #26
OSIG_OUTPUT27	Output #27
OSIG_OUTPUT28	Output #28
OSIG_OUTPUT29	Output #29
OSIG_OUTPUT30	Output #30
OSIG_OUTPUT31	Output #31
OSIG_OUTPUT32	Output #32
OSIG_OUTPUT33	Output #33
OSIG_OUTPUT34	Output #34
OSIG_OUTPUT35	Output #35
OSIG_OUTPUT36	Output #36
OSIG_OUTPUT37	Output #37
OSIG_OUTPUT38	Output #38
OSIG_OUTPUT39	Output #39
OSIG_OUTPUT40	Output #40
OSIG_OUTPUT41	Output #41
OSIG_OUTPUT42	Output #42
OSIG_OUTPUT43	Output #43
OSIG_OUTPUT44	Output #44
OSIG_OUTPUT45	Output #45
OSIG_OUTPUT46	Output #46
OSIG_OUTPUT47	Output #47
OSIG_OUTPUT48	Output #48
OSIG_OUTPUT49	Output #49

OSIG_OUTPUT50	Output #50
OSIG_OUTPUT51	Output #51
OSIG_OUTPUT52	Output #52
OSIG_OUTPUT53	Output #53
OSIG_OUTPUT54	Output #54
OSIG_OUTPUT55	Output #55
OSIG_OUTPUT56	Output #56
OSIG_OUTPUT57	Output #57
OSIG_OUTPUT58	Output #58
OSIG_OUTPUT59	Output #59
OSIG_OUTPUT60	Output #60
OSIG_OUTPUT61	Output #61
OSIG_OUTPUT62	Output #62
OSIG_OUTPUT63	Output #63
OSIG_RUNNING_GCODE	Gcode Running
OSIG_FEEDHOLD	Feed Hold
OSIG_BLOCK_DELETE	Block Delete
OSIG_SINGLE_BLOCK	Single Block
OSIG_REVERSE_RUN	Reverse Run
OSIG_OPT_STOP	Opt Stop
OSIG_MACHINE_ENABLED	Machine Enabled
OSIG_TOOL_CHANGE	Tool Change
OSIG_DIST_TOGO	Dist To Go
OSIG_MACHINE_CORD	Machine Coord
OSIG_SOFTLIMITS_ON	Softlimits On
OSIG_JOG_INC	Jog Inc
OSIG_JOG_CONT	Jog Cont
OSIG_JOG_ENABLED	Jog Enabled

OSIG_JOG MPG	Jog MPG
OSIG_HOMED_X	X Homed
OSIG_HOMED_Y	Y Homed
OSIG_HOMED_Z	Z Homed
OSIG_HOMED_A	A Homed
OSIG_HOMED_B	B Homed
OSIG_HOMED_C	C Homed
OSIG_DWELL	Dwell
OSIG_TP_MOUSE_DN	Toolpath Mouse Down
OSIG_LIMITOVER	Limit Override
OSIG_ALARM	Alarm
OSIG_PRTSF	Parts Finished

```

int mcSignalEnable(
    HMCSIG hSig, BOOL enabled);

rc = mc.mcSignalEnable(
    number hSig, number enabled)

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {

    m_cid = mInst; m_id = id;

    m_timer = new simTimer(this); m_cycletime = .001; HMCSIG hSig;

    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",
        DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);

    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,
        &m_Input0); mcIoRegister(m_hDev, "Output0", "Output0",
        IO_TYPE_OUTPUT, &m_Output0)

    if (mcSignalGetHandle(m_cid, ISIG_INPUT1, int sigid, &hSig) ==
        MERROR_NOERROR) {

        if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {

            <font color="green">// Signal mapped successfully</font> <font
            color="blue">mcSignalEnable(hSig, TRUE);</font> <font
            color="green">// Enable the signal.</font> }

        }
    }
}

```

```

int mcSignalGetHandle(
    MINSTANCE mInst, int sigid,
    HMCSIG *hSig);

hSig, rc = mc.mcSignalGetHandle(
    number mInst, number sigid)

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {
    m_cid = mInst; m_id = id;
    m_timer = new simTimer(this); m_cycletime = .001; HMCSIG hSig;
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",
        DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,
        &m_Input0); mcIoRegister(m_hDev, "Output0", "Output0",
        IO_TYPE_OUTPUT, &m_Output0)
    if (<font color="blue">mcSignalGetHandle(m_cid, ISIG_INPUT1,
        &hSig)</font> == MERROR_NOERROR) {
        if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {
            <font color="green">// Signal mapped successfully.</font>
            mcSignalEnable(hSig, true); <font color="green">// Enable the signal.
            </font> }
    }
}

```

```
int mcSignalGetInfo(  
    HMCSIG hSig, int *enabled, char *name, size_t namelen, char *desc,  
    size_t descLen, int *activeLow);  
  
enabled, name, desc, activeLow, rc = mc.mcSignalGetInfo(number hSig)  
  
HMCSIG hSig = 0;  
  
int enabled = 0;  
  
int nameLen = 20  
  
char nameBuf[nameLen]; int descLen = 40;  
  
char descBuf[descLen]; int activeLow = 0;  
  
  
  
while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcSignalGetInfo(hSig, &enabled, nameBuf,  
        nameLen, descBuf, descLen, &activeLow)</font>; } else {  
  
        break; }  
  
    }  
}
```

```
int mcSignalGetInfoStruct(  
    HMCSIG hSig, siginfo_t *siginf);  
  
siginf, rc = mc.mcSignalGetInfoStruct(  
    number hSig)  
  
struct siginfo {  
    char sigName[80]; char sigDesc[80]; int sigEnabled; int sigActiveLow;  
    HMCO sigMappedIoHandle; };  
  
typedef struct siginfo siginfo_t;  
  
HMCSIG hSig = 0;  
  
siginfo_t siginf;  
  
  
  
while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcSignalGetInfoStruct(hSig, &siginf);</font> } else  
{  
  
    break; }  
  
}  
}
```

```
int mcSignalGetNextHandle(  
    MINSTANCE mInst, int sigtype, HMCSIG startSig, HMCSIG *hSig);  
  
hSig, rc = mc.mcSignalGetNextHandle(  
    number mInst, number sigtype, number startSig)  
  
HMCSIG hSig = 0;  
  
  
  
while (<font color="blue">mcSignalGetNextHandle(0,  
    SIG_TYPE_INPUT, hSig, &hSig)</font> == MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="green">// Do something with hSig.</font> } else {  
  
        break; }  
  
    }  
}
```

```
int mcSignalGetState(  
    HMCSIG hSig, BOOL *state)  
  
state, rc = mc.mcSignalGetState(  
    number hSig)  
  
HMCSIG hSig = 0;  
  
BOOL sigState = FALSE;  
  
<font color="green">// Get the state of all input signals.</font> while  
(mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcSignalGetState(hSig, &sigState);</font> } else {  
  
        break; }  
  
}
```

```
int mcSignalHandleWait(  
    HMCSIG hSig, int waitMode, double timeoutSecs);  
  
rc = mc.mcSignalHandleWait(  
    number hSig, number waitMode, number timeoutSecs);  
  
MINSTANCE mInst = 0;  
  
HMCSIG hSig;  
  
int rc;  
  
rc = mcSignalGetHandle(mInst, ISIG_INPUT1 &hSig); if (rc ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcSignalHandleWait(hSig,  
WAIT_MODE_HIGH, .5)</font>; switch (rc) {  
  
        case MERROR_NOERROR: <font color="green">// Signal changed  
state from low to high</font> break; case MERROR_TIMED_OUT: <font  
color="green">// The signal didn't change state in the allotted time.</font>  
break; case MERROR_NOT_ENABLED: <font color="green">// The  
control was not enabled at the time of the function call or the control was  
disabled during the function call.</font> break; }  
  
}
```

```
int mcSignalMap(
    HMCSIG hSig, HMCIO hIo);
rc = mc.mcSignalMap(
    number hSig, number hIo)

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {
    m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime
    = .001; HMCSIG hSig;

    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);

    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,
    &m_Input0); mcIoRegister(m_hDev, "Output0", "Output0",
    IO_TYPE_OUTPUT, &m_Output0)

    if (mcSignalGetHandle(m_cid, ISIG_INPUT1, &hSig) ==
    MERROR_NOERROR) {

        if (<font color="blue">mcSignalMap(hSig, m_Input0)</font> ==
        MERROR_NOERROR) {

            <font color="green">// Signal mapped successfully</font> }

    }
}
```

```
int mcSignalSetActiveLow(
    HMCSIG hSig, BOOL activelow);
rc = mc.mcSignalSetActiveLow(
    number hSig, number activelow)
HMCSIG hSig = 0;
BOOL activeLow = TRUE;

<font color="green">// Set all input signals active low.</font> while
(mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig) ==
MERROR_NOERROR) {

    if (hSig != 0) {

        <font color="blue">mcSignalSetActiveLow(hSig, activeLow);</font> }
    else {

        break; }

}
```

```
int mcSignalSetState(  
    HMCSIG hSig, BOOL enabled);  
  
rc = mc.mcSignalSetState(  
    number hSig, number enabled)  
HMCSIG hSig = 0;  
  
  
  
<font color="green">// Set all output signals inactive.</font> while  
(mcSignalGetNextHandle(0, SIG_TYPE_OUTPUT, hSig, &hSig) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcSignalSetState(hSig, FALSE);</font> } else {  
  
        break; }  
  
}
```

```
int mcSignalUnmap(
    HMCSIG hSig);
rc = mc.mcSignalUnmap(
    number hSig)
MINSTANCE mInst = 0; HMCSIG hSig;
if (mcSignalGetHandle(mInst, ISIG_INPUT1, &hSig) ==
MERROR_NOERROR) {
    if (<font color="blue">mcSignalUnmap(hSig)</font> ==
MERROR_NOERROR) {
        <font color="green">// Signal now has no I/O mappings</font> }
    }
}
```

```
int mcSignalWait(  
    MINSTANCE mInst, int sigId, int waitMode, double timeoutSecs);  
  
rc = mc.mcSignalWait(  
    number mInst, number sigId, number waitMode, number timeoutSecs);  
  
MINSTANCE mInst = 0; HMCSIG hSig;  
  
int rc;  
  
  
  
rc = <font color="blue">mcSignalWait(mInst, ISIG_INPUT1,  
    WAIT_MODE_HIGH, .5)</font>; switch (rc) {  
  
    case MERROR_NOERROR: <font color="green">// Signal changed state  
        from low to high</font> break; case MERROR_TIMED_OUT: <font  
        color="green">// The signal didn't change state in the allotted time.</font>  
        break; case MERROR_NOT_ENABLED: <font color="green">// The  
        control was not enabled at the time of the function call or the control was  
        disabled during the function call.</font> break; }
```

Mach Registers

Mach registers are multi-purpose storage locations that can be defined (registered) by plugins. The parent to a register is a plugin device. Registers can contain data of any type, even string data.

Registers provide a unique storage location that is owned by the plugin that creates it. This is in contrast to Mach 3 DROs and LEDs that could be overwritten if two plugins wrote to the same DRO number not knowing that the other plugin also used that same DRO number. However, registers can be written and read from any script or plugin. But the name of the register has to be known!

The maximum string length is 4096 bytes for registers other than **REG_TYPE_COMMAND** type registers and 1024 bytes for the **REG_TYPE_COMMAND** type registers.

When a register is registered to the system, you must specify the register type. The different types are discussed below:

REG_TYPE_NONE specifies a register that is not typed. This kind of register is basically just a data container that will not notify any process if the data it stores changes.

REG_TYPE_INPUT specifies a register that can be mapped to a Mach input signal or an analog object. Mach and any GUI front end is notified if the data in this type of register is changed with a **MSG_REG_CHANGED** message. This type of register usually contains boolean data if mapped to a signal or ADC values.

REG_TYPE_INPUT_NUMERIC specifies a register that is the same as **REG_TYPE_INPUT** but can **ONLY** contain numeric data. Registers of this type typically can process faster than **REG_TYPE_INPUT** registers.

REG_TYPE_OUTPUT specifies a register that can be mapped to a Mach output signal or an analog object. The plugin that owns this register is

notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**.

REG_TYPE_OUTPUT_NUMERIC specifies a register that the same as **REG_TYPE_OUTPUT** but can **ONLY** contain numeric data. Registers of this type typically can process faster than **REG_TYPE_OUTPUT** registers. **REG_TYPE_HOLDING** specifies a general purpose register that cannot be mapped to either an input or output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**. This type of register usually contains boolean data.

REG_TYPE_COMMAND specifies a special purpose register that can be used to implement a simple command/response communication mechanism. Only the plugin that owns the register is notified with a **MSG_REG_COMMAND** synchronous message. It works with

- [mcRegSendCommand](#)
on the client side of the communication and
- [mcRegGetCommand](#)
and
- [mcRegSetResponse](#)
on the plugin side.

REG_TYPE_ENCODER specifies a register that can be mapped to a MPG or used to display the encoder counts in the GUI. Only Mach and the GUI are notified with a **MSG_REG_CHANGED** synchronous message.

- [mcRegGetCommand](#)
- [mcRegGetHandle](#)
- [mcRegGetInfo](#)
- [mcRegGetInfoStruct](#)
- [mcRegGetNextHandle](#)

- [mcRegGetUserData](#)
- [mcRegGetValue](#)
- [mcRegGetValueLong](#)
- [mcRegGetValueString](#)
- [mcRegGetValueStringClear](#)
- [mcRegRegister](#)
- [mcRegSendCommand](#)
- [mcRegSetDesc](#)
- [mcRegSetName](#)
- [mcRegSetResponse](#)
- [mcRegSetType](#)
- [mcRegSetUserData](#)
- [mcRegSetValue](#)
- [mcRegSetValueLong](#)
- [mcRegSetValueString](#)
- [mcRegUnregister](#)

```
int mcRegGetCommand(
```

```
    HMCREG hReg,
```

```
    char *cmd,
```

```
    size_t cmdLen)
```

N/A

```
<font color="green">// Retrieve the register command.</font> int  
simControl::ProcessMsg(long msg, long param1, long param2) {
```

```
    HMCREG hReg = (HMCREG)param1; long RegVal;
```

```
    switch (msg) {
```

```
        case MSG_REG_COMMAND: mcRegGetValueLong(hReg, &RegVal);  
        if (hReg = m_RegCommand) { <font color="green">// Is this our command  
        register?</font> char command[1024]; mcRegGetCommand(hReg,  
        command, sizeof(command)); wxString cmd(command);  
        cmd.MakeUpper(); if (cmd == wxT("THC ON")) {
```

```
            m_thc = true; mcRegSetResponse(hReg, "OK"); } else if (cmd ==  
            wxT("THC OFF")) {
```

```
            m_thc = false; mcMotionSync(m_cid); mcRegSetResponse(hReg, "OK");  
        } else if (cmd == wxT("THC STATUS")) {
```

```
            if (m_thc) {
```

```
                mcRegSetResponse(hReg, "1"); } else {
```

```
                mcRegSetResponse(hReg, "0"); }
```

```
        }
```

```
}
```

```
break;  
default:  
;  
}  
return(MERROR_NOERROR); }
```

```
int mcRegGetHandle(  
    MINSTANCE mInst, const char *path, HMCREG *hReg);  
  
hReg, rc = mc.mcRegGetHandle(  
    number mInst, string path)  
  
<font color="green">// Get the register handle.</font> MINSTANCE mInst  
= 0; HMCREG hReg;  
  
int rc;  
  
double value;  
  
if (<font color="blue">mcRegGetHandle(mInst, "core/global/Version",  
&hReg)</font> == MERROR_NOERROR) {  
  
    rc = mcRegGetValue(hReg, &value); }  
 
```

```
int mcRegGetInfo(
    HMCREG hReg, char *nameBuf, size_t nameBuflen, char *descBuf,
    size_t descBuflen, int *type, HMCDEV *hDev);

nameBuf, descBuf, type, hDev, rc = mc.mcRegGetInfo(
    number hReg)

HMCREG hReg = 0; char name[80];
char desc[80];
int type;
HMCDEV hDev;

while (mcSignalGetNextHandle(hDev, hReg, &hReg) ==
MERROR_NOERROR) {

    if (hSig != 0) {

        <font color="green">// Get the info on the register.</font>
mcRegGetInfo(hReg, name, sizeof(name), desc, sizeof(desc), &type,
&hDev); } else {

        break; }

}
```

```
int mcRegGetInfoStruct(  
    HMCREG hReg, reginfo_t *reginf);  
  
reginf, rc = mc.mcRegGetInfoStruct(  
    number hReg)  
  
struct reginfo {  
    char regName[80]; char regDesc[80]; int regType; HMCDEV regDev;  
    void *regUserData; int regInput; };  
  
typedef struct reginfo reginfo_t;  
  
HMCREG hReg = 0;  
  
reginfo_t reginf;  
  
  
  
while (mcSignalGetNextHandle(hDev, hReg, &hReg) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="green">// Get the info on the register.</font>  
        mcRegGetInfoStruct(hReg, ®inf); } else {  
  
        break; }  
  
    }  
}
```

```
int mcRegGetNextHandle(  
    HMCDEV hDev, HMCREG startReg, HMCREG *hReg);  
  
hReg, rc = mc.mcRegGetNextHandle(  
    number hDev, number startReg)  
  
HMCSIG hReg = 0;  
  
HMCDEV hDev;  
  
mcDeviceGetHandle(0, 0,&hDev); while (<font  
color="blue">mcRegisterGetNextHandle(hDev, hReg, &hReg)</font> ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="green">// do something with hReg.</font> } else {  
  
        break; }  
  
}
```

```
int mcRegGetUserData(  
    HMCREG hReg, void **data);
```

N/A

```
struct myDataStruct {  
    int myInt; char myString[80]; ...  
};
```

```
HMCREG hReg;
```

```
int mInst=0;
```

```
myDataStruct *data = NULL; void *dataPtr; <font color="green">// Get the  
handle to holding register 1 on SIM0.</font> if (mcRegGetHandle(mInst,  
"SIM0/HoldingReg1", &hReg) == MERROR_NOERROR) {
```

```
    mcRegGetUserData(hReg, &dataPtr); data = (myDataStruct *) dataPtr; }
```

```
int mcRegGetValue(  
    HMCREG hReg, double *value);  
  
value, rc = mcRegGetValue(  
    number hReg)  
  
<font color="green">// Get the register value.</font> MINSTANCE mInst  
= 0; HMCREG hReg; int rc;  
  
double value; if (mcRegGetHandle(mInst, "core/global/Version", &hReg)  
== MERROR_NOERROR) {  
  
    rc = mcRegGetValue(hReg, &value); }
```

```
int mcRegGetValueLong(  
    HMCREG hReg, double *value);  
  
value, rc = mcRegGetValueLong(  
    number hReg)  
  
<font color="green">// Get the register value.</font> MINSTANCE mInst  
= 0; HMCREG hReg; int rc;  
  
long value; if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = mcRegGetValueLong(hReg, &value); }
```

```
int mcRegGetValueString(
    HMCREG hReg, char *buf, size_t bufSize);

buf, rc = mcRegGetValueString(
    number hReg)

<font color="green">// Get the register value.</font> MINSTANCE mInst
= 0; HMCREG hReg;

int rc;

char value[80]; if (mcRegGetHandle(mInst, "core/global/Version", &hReg)
== MERROR_NOERROR) {

    rc = mcRegGetValueString(hReg, &value, sizeof(value)); }
```

```
int mcRegGetValueStringClear(
    HMCREG hReg, char *buf, size_t bufSize);

buf, rc = mcRegGetValueString(
    number hReg)

<font color="green">// Get the register value and clear it.</font>
MINSTANCE mInst = 0; HMCREG hReg;

int rc;

char value[80]; if (mcRegGetHandle(mInst, "core/global/Version", &hReg)
== MERROR_NOERROR) {

    rc = mcRegGetValueStringClear(hReg, &value, sizeof(value)); }
```

mcRegRegister

C/C++ Syntax:

```
int mcRegRegister(
    HMCDEV hDev,
    const char *regName,
    const char *regDesc,
    int regType,
    HMCREG *hReg);
```

LUA Syntax:

N/A

Description: Adds a register to the core.

Parameters:

Parameter	Description
hDev	The handle of the register's parent device.
regName	A string buffer specifying the name of the register.
regDesc	A string buffer specifying the description of the register.
regType	REG_TYPE_NONE, REG_TYPE_INPUT, REG_TYPE_OUTPUT, REG_TYPE_HOLDING, REG_TYPE_COMMAND, REG_TYPE_ENCODER.
hReg	The address of a HMCREG that receives the register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_DEVICE_NOT_FOUND	The hDev parameter was 0 or invalid.
MERROR_INVALID_ARG	regName , regDesc , or hReg is NULL.

Notes:

The **hDev** parameter **MUST** be a valid device handle. Otherwise, the function will crash.

Registers can contain data of any type, even string data. The maximum string length is 4096 bytes for registers other than **REG_TYPE_COMMAND** type registers and 1024 bytes for the **REG_TYPE_COMMAND** type registers.

REG_TYPE_NONE specifies a register that is not typed. This kind of register is basically just a data container that will not notify any process if the data it stores changes.

REG_TYPE_INPUT specifies a register that can be mapped to a Mach input signal. Mach and any GUI front end is notified if the data in this type of register is changed with a **MSG_REG_CHANGED** message. This type of register usually contains boolean data.

REG_TYPE_OUTPUT specifies a register that can be mapped to a Mach output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**.

REG_TYPE_HOLDING specifies a general purpose register that cannot be mapped to either an input or output signal. The plugin that owns this

register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**. This type of register usually contains boolean data.

REG_TYPE_COMMAND specifies a special purpose register that can be used to implement a simple command/response communication mechanism. Only the plugin that owns the register is notified with a **MSG_REG_COMMAND** synchronous message. It works with

- [mcRegSendCommand](#)
on the client side of the communication and
- [mcRegGetCommand](#)
and
- [mcRegSetResponse](#)
on the plugin side.

REG_TYPE_ENCODER specifies a register that can be mapped to a MPG or used to display the encoder counts in the GUI. Only Mach and the GUI are notified with a **MSG_REG_CHANGED** synchronous message.

Usage:

```
HMCREG m_hReg;  
HMCDEV m_hDev; // Contains the device handle from the device  
registration.  
int rc = mcRegRegister(m_hDev, "HoldingReg1", "HoldingReg1",  
REG_TYPE_HOLDING, &hReg);
```

```
int mcRegSendCommand(
    HMCREG hReg, const char *command, char *response, size_t
responseLen);

response, rc = mc.mcRegSendCommand(
    hReg, command)

<font color="green">// Send a command via a register.</font>
MINSTANCE mInst = 0;

if (mcRegGetHandle(mInst, "Sim0/SimCommand", &hReg) ==
MERROR_NOERROR) {

    char response[1024]; rc = <font
color="blue">mcRegSendCommand(hReg, "THC ON", response,
sizeof(response))</font>; if (rc == MERROR_NOERROR) {

        <font color="green">// check response.</font> }

    }
```

```
int mcRegSetDesc(  
    HMCREG hReg, const char *desc);  
  
rc = mc.mcRegSetDesc(  
    number hReg, string desc)  
  
<font color="green">// Set/change the description of the register.</font>  
MINSTANCE mInst = 0; if (mcRegGetHandle(mInst,  
"core/global/Version", &hReg) == MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetDesc(hReg, "Core Version")</font>; }  
}
```

```
int mcRegSetName(  
    HMCREG hReg, const char *name);  
  
rc = mc.mcRegSetName(  
    number hReg, string name)  
  
<font color="green">// Set/change the name of the register.</font>  
MINSTANCE mInst = 0; if (mcRegGetHandle(mInst,  
"core/global/Version", &hReg) == MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetName(hReg, "Version2")</font>;  
<font color="green">// The register path is now "core/global/Version2"  
</font> }
```

```
int mcRegSetResponse(
```

```
    HMCREG hReg,
```

```
    char *response);
```

N/A

```
<font color="green">// Set the response to a register command.</font> int
```

```
simControl::ProcessMsg(long msg, long param1, long param2) {
```

```
    HMCREG hReg = (HMCREG)param1; long RegVal;
```

```
    switch (msg) {
```

```
        case MSG_REG_COMMAND: mcRegGetValueLong(hReg, &RegVal);  
        if (hReg = m_RegCommand) { <font color="green">// Is this our command  
        register?</font> char command[1024]; mcRegGetCommand(hReg,  
        command, sizeof(command)); wxString cmd(command);  
        cmd.MakeUpper(); if (cmd == wxT("THC ON")) {
```

```
            m_thc = true; mcRegSetResponse(hReg, "OK"); } else if (cmd ==  
            wxT("THC OFF")) {
```

```
            m_thc = false; mcMotionSync(m_cid); mcRegSetResponse(hReg, "OK");  
            } else if (cmd == wxT("THC STATUS")) {
```

```
                if (m_thc) {
```

```
                    mcRegSetResponse(hReg, "1"); } else {
```

```
                    mcRegSetResponse(hReg, "0"); }
```

```
                }
```

```
            }
```

```
            break;
```

```
default:  
;  
}  
  
return(MERROR_NOERROR); }
```

```
int mcRegSetType(  
    HMCREG hReg, int regType);  
  
rc = mc.mcRegSetType(  
    number hReg, number regType)  
  
<font color="green">// Set the register type.</font> MINSTANCE mInst =  
0; if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetType(hReg, REG_TYPE_HOLDING)</font>; }  
}
```

```
mcRegSetUserData(  
    HMCREG hReg, void *data);  
  
MSG_REG_CHANGED  
  
HMCREG hReg;  
  
int mInst=0;  
  
RegisterStruct RegStruct; //User data is set to where the data struct is saved.  
  
<font color="green">// Get the handle to holding register 1 on SIM0.  
</font> mcRegGetHandle(mInst, "SIM0/HoldingReg1", &hReg);  
mcRegSetUserData(hReg, (void *)&RegStruct);
```

```
int mcRegSetValue(  
    HMCREG hReg, double value);  
  
rc = mc.mcRegSetValue(  
    number hReg, number value)  
  
<font color="green">// Set the register value.</font> MINSTANCE mInst =  
0; if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetValue(hReg, 4.0)</font>; }  
  
}
```

```
int mcRegSetValueLong(  
    HMCREG hReg, double value);  
  
rc = mc.mcRegSetValueLong(  
    number hReg, number value)  
  
<font color="green">// Set the register value.</font> MINSTANCE mInst =  
0;  
  
if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetValueLong(hReg, 4)</font>; }  
}
```

```
int mcRegSetValueString(  
    HMCREG hReg, const char *value);  
  
rc = mc.mcRegSetValueString(  
    number hReg, string value)  
  
<font color="green">// Set the register value.</font> MINSTANCE mInst =  
0;  
  
if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetValueString(hReg, "4.0")</font>; }  
}
```

```
int mcRegUnregister(  
    HMCDEV hDev, HMCREG hReg);
```

N/A

```
MINSTANCE mInst = 0; HMCREG hReg char *path = "Sim0/Register1";
```

```
mcRegGetHandle(mInst, path, &hReg); <font  
color="blue">mcRegUnregister(m_hDev, hReg);</font>
```

Mach Analog Objects

Mach analog objects form a layer that is superimposed over mapped registers. It is designed to allow for defining the register in terms of voltages instead of a raw numeric value

Motors

- [mcMotorGetAxis](#)
- [mcMotorGetCountsPerUnit](#)
- [mcMotorGetInfoStruct](#)
- [mcMotorGetMaxAccel](#)
- [mcMotorGetMaxVel](#)
- [mcMotorGetPos](#)
- [mcMotorGetVel](#)
- [mcMotorIsHomed](#)
- [mcMotorIsStill](#)
- [mcMotorRegister](#)
- [mcMotorSetHomePos](#)
- [mcMotorSetInfoStruct](#)
- [mcMotorSetMaxAccel](#)
- [mcMotorSetMaxVel](#)
- [mcMotorUnregister](#)

```
int mcMotorGetAxis(  
    MINSTANCE mInst, int motorId, int *axisId)  
  
axisId, rc = mc.mcMotorGetAxis(  
    number mInst, number motorId)  
  
<font color="green">// Get the parent axis for motor 0.</font>  
MINSTANCE mInst = 0; int axisId = -1;  
  
int rc = mcMotorGetAxis(mInst, 0, &axisId);
```

```
int mcMotorGetCountsPerUnit(  
    MINSTANCE mInst, int motorId, double *counts)  
counts, rc = mc.mcMotorGetCountsPerUnit(  
    number mInst, number motorId)  
  
<font color="green">// Get the counts per unit for motor 0.</font>  
MINSTANCE mInst = 0;  
  
double counts = 0.0;  
  
int rc = mcMotorGetCountsPerUnit(mInst, 0, &counts);
```

```
mcMotorGetInfoStruct(
```

```
    MINSTANCE mInst, int motorId,
```

```
    motorinfo_t *minf);
```

N/A

```
struct motorinfo {
```

```
    double CountsPerUnit; // Number of encoder counts or steps per unit.
```

```
    double MaxVelocity; // Max velocity of the axis.
```

```
    double MaxAcceleration; // Max rate to accelerate.
```

```
    BOOL Reverse; // Is the axis reversed?
```

```
    double BacklashAmount; // The amount of backlash in counts.
```

```
    double CurrentVelocity; // The speed the axis is moving, This could be  
    reported by the motion deivce.
```

```
    int CurrentPosition; // The Current Position (From the motion device).
```

```
    BOOL Homed; // True if the axis has been homed.
```

```
    long SoftMaxLimit; // Count for the max travel.
```

```
    long SoftMinLimit; // Count for the min travel.
```

```
    BOOL CanHome; // Can this motor home?
```

```
    BOOL Enabled; // Is this motor enabled?
```

```
    long EnableDelay; // ms to delay the enable signal for this motor.
```

```
    int AxisId; // -1 if no axis has mapped this motor.
```

```
};

typedef struct motorinfo motorinfo_t;

MINSTANCE mInst = 0;

int m = 2;

motorinfo_t minf;

mcMotorGetInfoStruct(mInst, m, &minf); <font color="green">// Get data
for motor 2.</font>
```

```
int mcMotorGetMaxAccel(  
    MINSTANCE mInst, int motorId, double *maxAccel);  
  
maxAccel, rc = mc.mcMotorGetMaxAccel(  
    number mInst, number motorId)  
  
<font color="green">// Get the max accel for motor 0.</font>  
MINSTANCE mInst = 0;  
  
double maxAccel = 0.0;  
  
int rc = mcMotorGetMaxAccel(mInst, 0, &maxAccel);
```

```
int mcMotorGetMaxVel(  
    MINSTANCE mInst, int motorId, double *maxVel)  
  
maxVel, rc = mc.mcMotorGetMaxVel(  
    number mInst, number motorId)  
  
<font color="green">// Get the max vel for motor 0.</font> MINSTANCE  
mInst = 0;  
  
double maxVel = 0.0;  
  
int rc = mcMotorGetMaxVel(mInst, 0, &maxVel);
```

```
int mcMotorGetPos(  
    MINSTANCE mInst, int motorId, double *val);  
  
val, rc = mc.mcMotorGetPos(  
    number mInst, number motorId)  
  
MINSTANCE mInst = 0; double Motor1Pos = 0; <font color="green">//  
Get the position of the Motor 1.</font> mcMotorGetPos(mInst, 1,  
&Motor1Pos);
```

```
int mcMotorGetVel(  
    MINSTANCE mInst, int motor, double *velocity);  
  
velocity, rc = mc.mcMotorGetVel(  
    number mInst, number motor)  
  
MINSTANCE mInst = 0; int m = MOTOR2;  
  
double CurrentVel = 0; mcMotorSetVel(mInst, m, &CurrentVel); <font  
color="green">// Get the current speed of motor2.</font>
```

```
int mcMotorIsHomed(  
    MINSTANCE mInst, int motorId, BOOL *homed);  
  
homed, rc = mc.mcMotorIsHomed(  
    number mInst, number motorId)  
  
MINSTANCE mInst = 0; int m = MOTOR2;  
  
int Homed = 0;  
  
mcMotorIsHomed(mInst, m, &Homed); <font color="green">// Get the  
homed state of motor2.</font>
```

```
int mcMotorIsStill(  
    MINSTANCE mInst, int motorId, BOOL *still);  
  
still, rc = mc.mcMotorIsStill(  
    number mInst, number motorId)  
  
MINSTANCE mInst = 0; int m = MOTOR0;  
  
int still = 0;  
  
mcMotorSetStill(mInst, m, &still); <font color="green">// Get the state if  
MOTOR0 is still</font>
```

mcMotorRegister(

 MINSTANCE mInst, int motorId);

N/A

 MINSTANCE mInst = 0; for (int m = MOTOR0; m < MOTOR4; m++) {

 mcMotorRegister(mInst, m);// Register motor0 -
 motor2 in the core. }

```
int mcMotorSetCountsPerUnit(  
    MINSTANCE mInst, int motorId, double counts);  
  
rc = mc.mcMotorSetCountsPerUnit(  
    number mInst, number motorId, number counts)  
  
<font color="green">// Set the counts for motor 0.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcMotorSetCountsPerUnit(mInst, 0, 1024);
```

```
int mcMotorSetHomePos(  
    MINSTANCE mInst, int motorId, int count);  
  
rc = mc.mcMotorSetHomePos(  
    number mInst, number motorId, number count)  
    MINSTANCE mInst = 0;  
  
int m = MOTOR2;  
  
int count = 1200;  
  
mcMotorSetHomePos(mInst, m, count); <font color="green">// Set the  
Home position of motor2.</font>
```

```
int mcMotorSetInfoStruct(  
    MINSTANCE mInst, int motoId,  
    motorinfo_t *minf);
```

N/A

```
struct motorinfo {  
    double CountsPerUnit; // Number of encoder counts or steps per unit.  
    double MaxVelocity; // Max velocity of the axis.  
    double MaxAcceleration; // Max rate to accelerate.  
    BOOL Reverse; // Is the axis reversed?  
    double BacklashAmount; // The amount of backlash in counts.  
    double CurrentVelocity; // The speed the axis is moving, This could be  
    // reported by the motion deivce.  
    int CurrentPosition; // The Current Position (From the motion device).  
    BOOL Homed; // True if the axis has been homed.  
    long SoftMaxLimit; // Count for the max travel.  
    long SoftMinLimit; // Count for the min travel.  
    BOOL CanHome; // Can this motor home?  
    BOOL Enabled; // Is this motor enabled?  
    long EnableDelay; // ms to delay the enable signal for this motor.  
};
```

```
typedef struct motorinfo motorinfo_t;

MINSTANCE mInst = 0;

int m = MOTOR2;

motorinfo_t minf;

mcMotorGetInfoStruct(mInst, m, &minf); <font color="green">// Get data
for motor2.</font> minf.CountsPerUnit /= 2;<font color="green">//Divid
motor counts per unit by 2.</font> mcMotorSetInfoStruct(mInst, m,
&minf); <font color="green">// Set data for motor2.</font>
```

```
int mcMotorSetMaxAccel(  
    MINSTANCE mInst, int motorId, double maxAccel);  
  
rc = mc.mcMotorSetMaxAccel(  
    number mInst, number motorId, number maxAccel)  
  
<font color="green">// Set a new accel value for motor 0.</font>  
MINSTANCE mInst = 0;  
  
int mcMotorSetMaxAccel(mInst, 0, 75);
```

```
int mcMotorSetMaxVel(  
    MINSTANCE mInst, int motorId, double maxVel);  
  
rc = mc.mcMotorSetMaxVel(  
    number mInst, number motorId, number maxVel)  
  
<font color="green">// Set the maximum velocity for motor 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMotorSetMaxVel(mInst, 0, 500);
```

```
int mcMotorUnregister(
```

```
    MINSTANCE mInst, int motorId);
```

N/A

```
MINSTANCE mInst = 0; int m = MOTOR2; mcMotorUnregister(mInst,  
m);<font color="green">// Unregister motor2 from the Core.</font>
```

```
MINSTANCE mInst = 0;

int rc = mcMotorMapSetDefinition(mInst, 0, <font color="green">// Motor
ID.</font> 480000, <font color="green">// Length or screw in counts.
</font> 48); <font color="green">// Number of measured points.</font>

rc = mcMotorMapSetPoint(mInst,
0, <font color="green">// Our first measured point.</font> 4); <font
color="green">// The error, in counts.</font>
```

Repeat the same procedure for every point ending at point 23. All points will need to be measured.

Measuring at random intervals is not supported. The measuring points are evenly spread across the length of the screw. The points in between the measuring points are interpolated to spread the error across the measurement distance. There is no limit to the number of points a map can contain and increasing that number will not slow the system down. About the only impact the number of measuring points will have is in the measurement process itself. More measuring points will of course yield finer resolution but the user may reach a point of diminishing returns.

- [mcMotorMapGetDefinition](#)
- [mcMotorMapGetLength](#)
- [mcMotorMapGetPoint](#)
- [mcMotorMapGetPointCount](#)
- [mcMotorMapGetStart](#)
- [mcMotorMapSetDefinition](#)
- [mcMotorMapSetLength](#)
- [mcMotorMapSetPoint](#)
- [mcMotorMapSetPointCount](#)
- [mcMotorMapSetStart](#)

```
int mcMotorMapGetDefinition(  
    MINSTANCE mInst, int motorId, long *lengthCounts, long *numPoints);  
  
lengthCounts, numPoints, rc = mc.mcMotorMapGetDefinition(  
    number mInst, number motorId)  
  
<font color="green">// </font> MINSTANCE mInst = 0;  
  
long lengthCounts;  
  
long numPoints;  
  
int rc = mcMotorMapGetDefinition(mInst, 0, &lengthCounts,  
    &numPoints);
```

```
int mcMotorMapGetLength(  
    MINSTANCE mInst, int motorId, int *length);  
  
length, rc = mcMotorMapGetLength(  
    number mInst, number motorId)  
  
<font color="green">// Get the screw length for motor zero.</font>  
MINSTANCE mInst = 0;  
  
int length;  
  
int rc = mcMotorMapGetLength(mInst, 0, &length);
```

```
int mcMotorMapGetPoint(  
    MINSTANCE mInst, int motorId, int point, int *error);  
  
error, rc = mc.mcMotorMapGetPoint(  
    numbser mInst, numbser motorId, numbser point)  
  
<font color="green">// Get the screw map error for motor 0, measurement  
point 10.</font> MINSTANCE mInst = 0;  
  
int screwErr;  
  
int rc = mcMotorMapGetPoint(mInst, 0, 10, &screwErr);
```

```
int mcMotorMapGetPointCount(  
    MINSTANCE mInst, int motorId, int *points);  
  
points, rc = mc.mcMotorMapGetPointCount(  
    number mInst, number motorId)  
  
<font color="green">// Get the number of measurement points.</font>  
MINSTANCE mInst = 0;  
  
int points;  
  
int rc = mcMotorMapGetPointCount(mInst, 0, &points);
```

```
int mcMotorMapGetStart(  
    MINSTANCE mInst, int motorId, int *startPoint);  
  
startPoint, rc = mc.mcMotorMapGetStart(  
    number mInst, number motorId)  
  
<font color="green">// Get the starting point for the motor 0 screw map.  
</font> MINSTANCE mInst = 0;  
  
int startPoint;  
  
int rc = mcMotorMapGetStart(mInst, 0, &startPoint);
```

```
int mcMotorMapSetDefinition(  
    MINSTANCE mInst, int motorId, long lengthCounts, long numPoints);  
  
rc = mc.mcMotorMapSetDefinition(  
    number mInst, number motorId, number lengthCounts, number  
    numPoints)  
  
<font color="green">// Set the screw map definition for motor 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetDefinition(mInst, 0, 20000, 20);
```

```
int mcMotorMapSetLength(  
    MINSTANCE mInst, int motorId, int length);  
  
rc = mc.mcMotorMapSetLength(  
    number mInst, number motorId, number length);  
  
<font color="green">// Set the screw length for the motor 0 screw map.  
</font> MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetLength(mInst, 0, 20000);
```

```
int mcMotorMapSetPoint(  
    MINSTANCE mInst, int motorId, int point, int error);  
  
rc = mc.mcMotorMapSetPoint(  
    number mInst, number motorId, number point, number error);  
  
<font color="green">// Set the error in motor 0 screw map.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetPoint(mInst, 0, 5, 12);
```

```
int mcMotorMapSetPointCount(  
    MINSTANCE mInst, int motorId, int points);  
  
rc = mc.mcMotorMapSetPointCount(  
    number mInst, number motorId, number points);  
  
<font color="green">// Set the number of measured points in the screw map  
for motor 0.</font> MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetPointCount(mInst, 0, 20);
```

```
int mcMotorMapSetStart(  
    MINSTANCE mInst, int motorId, int start);  
  
rc = mc.mcMotorMapSetStart(  
    number mInst, number motorId, number start)  
  
<font color="green">// Set the starting point number for the motor 0 screw  
map. (usually 0)</font> MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetStart(mInst, 0, 0);
```



Home



Up a level



Previous



Next

Axes

- [mcAxisDeref](#)
- [mcAxisDerefAll](#)
- [mcAxisEnable](#)
- [mcAxisGetHomeDir](#)
- [mcAxisGetHomeInPlace](#)
- [mcAxisGetHomeOffset](#)
- [mcAxisGetHomeOrder](#)
- [mcAxisGetHomeSpeed](#)
- [mcAxisGetInfoStruct](#)
- [mcAxisGetMachinePos](#)
- [mcAxisGetMotorId](#)
- [mcAxisGetOverrideAxis](#)
- [mcAxisGetPos](#)
- [mcAxisGetProbePos](#)
- [mcAxisGetProbePosAll](#)
- [mcAxisGetScale](#)

- [mcAxisGetSoftlimitEnable](#)
- [mcAxisGetSoftlimitMax](#)
- [mcAxisGetSoftlimitMin](#)
- [mcAxisGetSpindle](#)
- [mcAxisGetVel](#)
- [mcAxisHome](#)
- [mcAxisHomeAll](#)
- [mcAxisHomeComplete](#)
- [mcAxisHomeCompleteWithStatus](#)
- [mcAxisIsEnabled](#)
- [mcAxisIsHomed](#)
- [mcAxisIsStill](#)
- [mcAxisMapMotor](#)
- [mcAxisRegister](#)
- [mcAxisRemoveOverrideAxis](#)
- [mcAxisRemoveOverrideAxisSync](#)
- [mcAxisSetHomeDir](#)
- [mcAxisSetHomeInPlace](#)
- [mcAxisSetHomeOffset](#)
- [mcAxisSetHomeOrder](#)

- [mcAxisSetHomeSpeed](#)
- [mcAxisSetInfoStruct](#)
- [mcAxisSetMachinePos](#)
- [mcAxisSetOverrideAxis](#)
- [mcAxisSetPos](#)
- [mcAxisSetSoftlimitEnable](#)
- [mcAxisSetSoftlimitMax](#)
- [mcAxisSetSoftlimitMin](#)
- [mcAxisSetSpindle](#)
- [mcAxisSetVel](#)
- [mcAxisUnmapMotor](#)
- [mcAxisUnmapMotors](#)
- [mcAxisUnregister](#)

```
int mcAxisDeref(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisDeref(  
    number mInst, number axisId)  
  
int mInst = 0;  
  
<font color="green">// Set AXIS0 to deref (could have used X_AXIS).  
</font> mcAxisDeref(mInst, AXIS0);
```

```
int mcAxisDerefAll(  
    MINSTANCE mInst);  
  
rc = mc.mcAxisDerefAll(  
    number mInst)  
  
int mInst = 0;  
  
<font color="green">// Set all axis to deref.</font>  
mcAxisDerefAll(mInst);
```

```
int mcAxisEnable(  
    MINSTANCE mInst, int axisId, BOOL enabled);  
  
rc = mc.mcAxisEnable(  
    number mInst, number axisId, number enabled);  
  
int mInst = 0;  
  
<font color="green">// Set Y_AXIS to be disabled.</font>  
mcAxisEnable(mInst, Y_AXIS, false);
```

```
int mcAxisGetHomeDir(  
    MINSTANCE mInst, int axisId, int *dir);  
  
dir, rc = mc.mcAxisGetHomeDir(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int dir = 0;  
  
<font color="green">// Get Home offset of the X axis.</font>  
mcAxisGetHomeDir(mInst, X_AXIS, &dir);
```

```
int mcAxisGetHomeInPlace(  
    MINSTANCE mInst, int axisId, BOOL *homeInPlace);  
  
homeInPlace, rc = mc.mcAxisGetHomeInPlace(  
    number mInst, number axisId)  
  
<font color="green">// Get the Home In Place flag for the X axis.</font>  
MINSTANCE mInst = 0;  
  
BOOL hip = FALSE;  
  
mcAxisGetHomeInPlace(mInst, X_AXIS, &hip);
```

```
int mcAxisGetHomeOffset(  
    MINSTANCE mInst, int axisId, double *offset);  
  
offset, rc = mc.mcAxisGetHomeOffset(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int offset = 0;  
  
<font color="green">// Get Home offset of the X axis.</font>  
mcAxisGetHomeOrder(mInst, X_AXIS, &offset);
```

```
int mcAxisGetHomeOrder(  
    MINSTANCE mInst, int axisId, int *order);  
  
order, rc = mc.mcAxisGetHomeOrder(  
    number mInst, number axisId)  
  
int mInst=0;  
  
double XAxisPos = 0;  
  
int order = 0;  
  
<font color="green">// Get Home order of the X axis.</font>  
mcAxisGetHomeOrder(mInst, X_AXIS, &order);
```

```
int mcAxisGetHomeSpeed(  
    MINSTANCE mInst, int axisId, double *percent);  
  
percent, rc = mc.mcAxisGetHomeSpeed(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int rc = 0;  
  
double XAxisPos = 0; double percent = 0; double maxVel = 0  
  
double homeVel = 0;  
  
<font color="green">// Get home speed percentage of the X axis.</font>  
mcAxisGetHomeSpeed(mInst, X_AXIS, &percent); mcAxisGetVel(mInst,  
X_AXIS, &maxVel); homeVel = maxVel * percent;
```

```
int mcAxisGetInfoStruct(  
    MINSTANCE mInst,  
    int axisId,  
    axisinfo_t *ainf);  
  
ainf, rc = mc.mcAxisGetInfoStruct(number mInst, number axisId)  
  
struct axisinfo {  
  
    BOOL OutOfBandAxis; // Is this an out of band axis?  
  
    BOOL IsStill; // Set high when the axis is not moving BOOL Jogging; //  
    Used to tell to jog...  
  
    BOOL Homing; // Used to tell the state of the home operation.  
  
    int Id; // Axis Id  
  
    BOOL IsSpindle; // Does this axis control a spindle?  
  
    BOOL Enabled; // Is axis enabled?  
  
    BOOL SoftlimitEnabled; // Softlimits enabled?  
  
    double SoftMaxLimit; // Count for the max travel.  
  
    double SoftMinLimit; // Count for the m.  
  
    BOOL VelocityMode; // Used to make the axis move at a fixed speed  
    BOOL BufferJog; // Buffer all jogs?  
  
    double Pos; // Position in user units.  
  
    double Mpos; // Machine position in user units.  
  
    int HomeOrder; // The order in which to home the axis.
```

```
int HomeDir; // The direction the axis homes.

double HomeOffset; // The offset from the the limits switch.

double HomeSpeedPercent;// The percentage of the max velocity at
which to home.

BOOL SoftlimitUsed; // Use Softlimits?

BOOL HomeInPlace; // Zero the axis in place when Refed?

int MotorId[MC_MAX_AXIS_MOTORS]; //child motor ID array.

};

typedef struct axisinfo axisinfo_t;

int mInst = 0;

axisinfo_t ainf;

<font color="green">// Get Y_AXIS info structure.</font>
mcAxisGetInfoStruct(mInst, Y_AXIS, &ainf);
```

```
int mcAxisGetMachinePos(  
    MINSTANCE mInst, int axisId, double *val);  
  
val, rc = mc.mcAxisGetMachinePos(  
    number mInst, number axisId)  
  
int mInst=0;  
  
double XAxisMachinePos = 0; int AxisNumber = X_AXIS;  
  
mcAxisGetMachinePos(mInst, AxisNumber, &XAxisMachinePos); <font  
color="green">// Get the position of the X axis in Machine Pos.</font>
```

```
int mcAxisGetMotorId(  
    MINSTANCE mInst, int axis, int childId, int *motorId);  
  
motorId, rc = mc.mcAxisGetMotorId(  
    number mInst, number axis, number childId)  
  
int mInst=0;  
  
int motorIds[]={-1,-1,-1,-1,-1}  
  
int id;  
  
<font color="green">// Get all the motor ID's for the X axis up to 5 motors.  
</font> for(int i = 0; i < 5; i++){  
  
    if (mcAxisGetMotorId(mInst, X_AXIS, i, &id) ==  
  
        MERROR_NOERROR No Error.) {  
  
        motorIds[i] = id; }  
  
    }  
}
```

```
int mcAxisGetOverrideAxis(  
    MINSTANCE mInst, int axis, int *axis1, int *axis2, int *axis3, int  
    *axis4);  
  
axis1, axis2, axis3, axis4, rc = mc.mcAxisGetOverrideAxis(  
    number mInst, number axis)  
  
int mInst = 0;  
  
int ora[4];  
  
<font color="green">// Get the override axis for the Z axis.</font>  
mcAxisGetOverrideAxis(mInst, ZAXIS, &ora[0], &ora[1], &ora[2],  
    &ora[3]);
```

```
int mcAxisGetPos(  
    MINSTANCE mInst, int axisId, double *val);  
  
val, rc = mc.mcAxisGetPos(  
    number mInst, number axisId)  
  
int mInst=0;  
  
double XAxisPos = 0;  
  
int AxisNumber = X_AXIS; <font color="green">// Get the position of the  
X axis.</font> mcAxisGetPos(mInst, AxisNumber, &XAxisPos);
```

```
int mcAxisGetProbePos(  
    MINSTANCE mInst, int axisId, BOOL machinePos, double *val);  
  
val, rc = mc.mcAxisGetProbePos(  
    number mInst, number axisId, number machinePos)  
  
<font color="green">// Get the last probe strike position for the X  
axis</font> MINSTANCE mInst = 0;  
  
double xProbePos = 0;  
  
mcAxisGetProbePos(mInst, X_AXIS, FALSE, &xProbPos);
```

```
int mcAxisGetProbePosAll(  
    MINSTANCE mInst, BOOL machinePos, double *x, double *y, double  
    *z, double *a, double *b, double *c);  
  
x, y, z, a, b, c, rc = mc.mcAxisGetProbePosAll(  
    number mInst, number machinePos)  
  
<font color="green">// Get the last probe strike position for the X, Y, and Z  
axes.</font> MINSTANCE mInst = 0;  
  
double x, y, z;  
  
int rc = mcAxisGetProbePosAll(mInst, FALSE, &x, &y, &z, NULL,  
NULL, NULL); if (rc == MERROR_NOERROR) {  
  
    // Process probe positions...  
  
}
```

```
int mcAxisGetScale(  
    MINSTANCE mInst, int axisId, double *scaleVal);  
  
scaleVal, rc = mc.mcAxisGetProbePos(  
    number mInst, number axisId)  
  
<font color="green">// Get the scale value for the X axis</font>  
MINSTANCE mInst = 0; double scale = 0;  
  
mcAxisGetScale(mInst, X_AXIS, &scale);
```

```
int mcAxisGetSoftlimitEnable(  
    MINSTANCE mInst, int axisId, int *enable);  
  
enable, rc = mc.mcAxisGetSoftlimitEnable(  
    number mInst, number axisId)  
  
<font color="green">// Get master soft limit enable state for axis 0.</font>  
int mInst = 0;  
  
int enable = 0;  
  
mcAxisGetSoftlimitEnable(mInst, 0, &enable);
```

```
int mcAxisSetSoftlimitMax(  
    MINSTANCE mInst, int axisId, double max);  
  
rc = mc.mcAxisSetSoftlimitMax(  
    number mInst, number axisId, number max);  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double max = 20;  
  
mcAxisGetSoftlimitMax( mInst, axis, max);<font color="green">// Set the  
max distance of the softlimit for the Y axis to 20 units.</font>
```

```
int mcAxisSetSoftlimitMin(  
    MINSTANCE mInst, int axisId, double min);  
  
rc = mc.mcAxisSetSoftlimitMin(  
    number mInst, number axisId, number min);  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double min = 20;  
  
mcAxisGetSoftlimitMin( mInst, axis, min);<font color="green">// Set the  
min distance of the softlimit for the Y axis to 20 units.</font>
```

```
int mcAxisGetSpindle(  
    MINSTANCE mInst, int axisId, bool *spindle);  
  
spindle, rc = mcAxisGetSpindle(  
    number mInst, number axisId)  
  
<font color="green">// Find the spindle axis</font> int mInst = 0;  
int axisId;  
  
bool isSpindle = false; for (axisId = MC_MAX_COORD_AXES; axisId <  
MC_MAX_AXES; axisId++) {  
  
    mcAxisGetSpindle(mInst, axisId, &isSpindle); if (isSpindle) {  
  
        <font color="green">// Spindle found!</font> break; }  
  
    }  
}
```

```
int mcAxisGetVel(  
    MINSTANCE mInst, int axisId, double *velocity);  
  
velocity, rc = mc.mcAxisGetVel(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double CurrentVel = 0; mcAxisGetVel(mInst, axis, &CurrentVel); <font  
color="green">// Get the current speed of the Y axis.</font>
```

```
int mcAxisHome(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisHome(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; <font color="green">// Set the X axis to home.  
</font> mcAxisSetPos(mInst, AxisNumber);
```

```
int mcAxisHomeAll(  
    MINSTANCE mInst);  
  
rc = mc.mcAxisHomeAll(  
    number mInst)  
  
int mInst=0;  
  
<font color="green">// Home all coordinated axes.</font>  
mcAxisHomeAll(mInst);
```

```
int mcAxisHomeComplete(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisHomeComplete(  
    number mInst, number axisId);  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; <font color="green">// Mark the X axis home  
operation as complete.</font> mcAxisHomeComplete(mInst,  
AxisNumber);
```

```
int mcAxisHomeCompleteWithStatus(  
    MINSTANCE mInst, int axisId BOOL success);  
  
rc = mc.mcAxisHomeCompleteWithStatus(  
    number mInst, number axisId number success);  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; <font color="green">// Mark the X axis  
homing operation as finished and that it completed with success.</font>  
mcAxisHomeCompleteWithStatus(mInst, AxisNumber, TRUE);
```

```
int mcAxisIsEnabled(  
    MINSTANCE mInst, int axisId, BOOL *enabled);  
  
enabled, rc = mc.mcAxisIsEnabled(  
    number mInst, number axisId)  
  
<font color="green">// Find all enabled axes.</font> int mInst = 0;  
  
int axisId;  
  
BOOL enabled = FALSE;  
  
for (axisId = 0; axisId < MC_MAX_AXES; axisId++) {  
  
    enabled = false; mcAxisIsEnabled(mInst, axisId, &enabled); if (enabled  
    == TRUE) {  
  
        <font color="green">// Axis is enabled!</font> }  
  
    }  
}
```

```
int mcAxisIsHomed(  
    MINSTANCE mInst, int axisId, BOOL *homed);  
  
homed, rc= mc.mcAxisIsHomed(  
    number mInst, number axisId)  
  
int mInst = 0;  
  
BOOL homed = FALSE; <font color="green">// Get the homed state of Z  
axis.</font> mcAxisIsHomed(mInst, Z_AXIS, &homed); if (rc ==  
MERROR_NOERROR && homed == TRUE) {  
  
    <font color="green">// Z is homed.</font> }
```

```
int mcAxisIsHomin(
    MINSTANCE mInst, int axisId, BOOL *homing);
    homing, rc= mc.mcAxisIsHomed(
        number mInst, number axisId)
    int mInst = 0;
    BOOL homing = FALSE; <font color="green">// Get the homing state of Z
    axis.</font> mcAxisIsHoming(mInst, Z_AXIS, &homing); if (rc ==
    MERROR_NOERROR && homing == TRUE) {
        <font color="green">// Z is in a home operation.</font> }
```

```
int mcAxisIsStill(  
    MINSTANCE mInst, int axisId, BOOL *still);  
  
still, rc = mc.mcAxisIsStill(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; int still = 0;  
  
<font color="green">// Get the is moving state of the axis vluue of 1 is  
stopped.</font> mcAxisIsStill(mInst, AxisNumber, &still);
```

```
int mcAxisMapMotor(  
    MINSTANCE mInst, int axisId, int motorId);  
  
rc = mc.mcAxisMapMotor(  
    number mInst, number axisId, number motorId)  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; int MotorNumber = MOTOR5  
  
<font color="green">// Map motor5 to the X axis.</font>  
mcAxisMapMotor(mInst, AxisNumber, MotorNumber);
```

```
int mcAxisRegister(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisRegister(  
    number mInst, number axisId);  
  
int mInst=0;  
  
for (int a = AXIS0; < AXIS4; a++) {  
    <font color="green">// Register axis0 - axis2 to the core.</font> if(  
    mcAxisRegister(mInst, a) !=  
  
    MERROR_NOERROR No Error.){  
        return(false); }  
    }  
}
```

```
int mcAxisRemoveOverrideAxis(  
    MINSTANCE mInst, int axisId, int axisToOverrideId);  
  
rc = mc.mcAxisRemoveOverrideAxis(  
    number mInst, number axisId, number axisToOverrideId)  
  
int mInst = 0;  
  
int homed = 0;  
  
<font color="green">// Remove axis 8 from the Z axis.</font>  
mcAxisRemoveOverrideAxis( mInst, AXIS_8, ZAXIS );
```

```
int mcAxisRemoveOverrideAxisSync(  
    MINSTANCE mInst, int axisId,  
    int axisToOverrideId);  
  
rc = mc.mcAxisRemoveOverrideAxisSync(  
    number mInst, number axisId, number axisToOverrideId)  
  
int mInst = 0;  
  
int homed = 0;  
  
<font color="green">// Remove axis 8 from the Z axis.</font>  
mcAxisRemoveOverrideAxisSync( mInst, AXIS_8, ZAXIS );
```

```
int mcAxisSetHomeDir(  
    MINSTANCE mInst, int axisId, int dir);  
  
rc = mc.mcAxisSetHomeDir(  
    number mInst, number axisId, number dir)  
  
<font color="green">// Set axis 0 homing directio to POS.</font> int mInst  
= 0;  
  
mcAxisSetHomeDir(mInst, 0, 1);
```

```
int mcAxisSetHomeInPlace(  
    MINSTANCE mInst, int axisId, BOOL homeInPlace);  
  
rc = mc.mcAxisGetHomeInPlace(  
    number mInst, number axisId, number homeInPlace)  
  
<font color="green">// Set the Home In Place flag for the X axis.</font>  
MINSTANCE mInst = 0;  
  
BOOL hip = FALSE;  
  
mcAxisGetHomeInPlace(mInst, X_AXIS, hip);
```

```
int mcAxisSetHomeOffset(  
    MINSTANCE mInst, int axisId, double offset);  
  
rc = mcAxisSetHomeOffset(  
    number mInst, number axisId, number offset);  
  
<font color="green">// Set the home offset for axis 0.</font> int mInst = 0;  
  
mcAxisSetHomeOffset(mInst, 0, 1.5 <font color="green"> /* inches  
 */</font>);
```

```
int mcAxisSetHomeOrder(  
    MINSTANCE mInst, int axisId, int order);  
  
rc = mc.mcAxisSetHomeOrder(  
    number mInst, number axisId, number order)  
  
int mInst=0;  
  
<font color="green">// Set the order of homming so the Z axis will home  
first then the X and Y.</font> mcAxisSetHomeOrder(mInst, Z_AXIS , 0);  
mcAxisSetHomeOrder(mInst, X_AXIS , 1); mcAxisSetHomeOrder(mInst,  
Y_AXIS , 1);
```

```
int mcAxisSetHomeSpeed(  
    MINSTANCE mInst, int axis, double percent);  
  
rc = mc.mcAxisSetHomeSpeed(  
    number mInst, number axis, number percent);  
  
<font color="green">// Set the homing speed for axis 0 as a percentage of  
the max velocity.</font> int mInst = 0;  
  
mcAxisSetHomeSpeed(mInst, 0, 20.5 <font color="green">/* percent  
*/*</font>);
```

```
int mcAxisSetInfoStruct(
```

```
    MINSTANCE mInst, int axisID,
```

```
    axisinfo_t *ainf);
```

N/A

```
struct axisinfo {
```

```
    bool OutOfBandAxis; bool IsStill; // Set high when the axis is not  
    moving int Jogging; // Used to tell to jog...
```

```
    int Homing; // Used to tell the state of the home operation.
```

```
    int Id; // Axis Id bool IsSpindle; // Does this axis control a spindle?
```

```
    bool Enabled; // Is axis enabled?
```

```
    bool SoftlimitEnabled; // Softlimits enabled?
```

```
    double SoftMaxLimit; // Count for the max travel.
```

```
    double SoftMinLimit; // Count for the min travel.
```

```
    bool VelocityMode; // Used to make the axis move at a fixed speed  
    bool BufferJog; double Pos; // Position in user units.
```

```
    double Mpos; // Machine position in user units.
```

```
    int HomeOrder; // The order in which to home the axis.
```

```
};
```

```
int mInst = 0;
```

```
axisinfo_t ainf;
```

```
mcAxisGetInfoStruct(mInst, Y_AXIS, &ainf);<font color="green">// Get
Y_AXIS info structure.</font> ainf.HomeOrder = 1;<font color="green">//
Set Y_AXIS home order to 1.</font> mcAxisSetInfoStruct(mInst,
Y_AXIS, &ainf);<font color="green">// Set Y_AXIS info structure.</font>
```

```
int mcAxisSetMachinePos(  
    MINSTANCE mInst, int axis, double val);  
  
rc = mc.mcAxisSetMachinePos(  
    number mInst, number axis, number val)  
  
<font color="green">// Set axis 0 fisxture offset.</font> int mInst = 0;  
  
mcAxisSetMachinePos(mInst, 0, 0.0 <font color="green"> /* set part zero  
 */</font>);
```

```
int mcAxisSetOverrideAxis(  
    MINSTANCE mInst, int axisTd, int axisToOverrideId);  
  
rc = mc.mcAxisSetOverrideAxis(  
    number mInst, number axisId, number axisToOverrideId)  
  
int mInst = 0;  
  
int homed = 0;  
  
<font color="green">// Set the axis 8 to be the override axis for the Z axis.  
</font> mcAxisSetOverrideAxis(mInst, AXIS_8, ZAXIS);
```

```
int mcAxisSetPos(  
    MINSTANCE mInst, int axisId, double val);  
  
rc = mc.mcAxisSetPos(  
    number mInst, number axisId, number val);  
  
int mInst=0;  
  
double XAxisPos = .5;  
  
int AxisNumber = X_AXIS; <font color="green">// Set the position of the  
X axis by changing the fixture offset.</font> mcAxisSetPos(mInst,  
AxisNumber, XAxisPos);
```

```
int mcAxisSetSoftlimitEnable(  
    MINSTANCE mInst, int axis, int enabel);  
  
rc = mc.mcAxisSetSoftlimitEnable(  
    number mInst, number axis, number enabel)  
  
<font color="green">// Diable softlimits on axis 0</font> MINSTANCE  
mInst = 0;  
  
mcAxisSetSoftlimitEnable(mInst, 0,0);
```

```
int mcAxisSetSoftlimitMax(  
    MINSTANCE mInst, int axisId, double max);  
  
rc = mc.mcAxisSetSoftlimitMax(  
    number mInst, number axisId, number max);  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double max = 20;  
  
mcAxisGetSoftlimitMax( mInst, axis, max);<font color="green">// Set the  
max distance of the softlimit for the Y axis to 20 units.</font>
```

```
int mcAxisSetSoftlimitMin(  
    MINSTANCE mInst, int axisId, double min);  
  
rc = mc.mcAxisSetSoftlimitMin(  
    number mInst, number axisId, number min);  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double min = 20;  
  
mcAxisGetSoftlimitMin( mInst, axis, min);<font color="green">// Set the  
min distance of the softlimit for the Y axis to 20 units.</font>
```

```
int mcAxisSetSpindle(  
    MINSTANCE mInst, int axisId, BOOL spindle);  
  
rc = mc.mcAxisSetSpindle(  
    number mInst, number axisId, number spindle)  
  
<font color="green">// Set axis 6 as a spindle axis.</font> MINSTANCE  
mInst = 0;  
  
mcAxisSetSpindle(mInst, 6, true);
```



Home



Up a level



Previous



Next

mcAxisSetVel

C/C++ Syntax:

```
int mcAxisSetVel(  
    MINSTANCE mInst,  
    int axis,  
    double velocity);
```

Description: Not used at this time.

```
int mcAxisUnmapMotor(  
    MINSTANCE mInst, int axisId, int motor);  
  
rc = mc.mcAxisUnmapMotor(  
    number mInst, number axisId, number motor)  
  
int mInst = 0;  
  
<font color="green">// Remove motor6 from the Y axis.</font>  
mcAxisUnmapMotor(mInst, Y_AXIS, MOTOR6);
```

```
int mcAxisUnmapMotors(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisUnmapMotors(  
    number mInst, number axisId)  
  
int mInst = 0;  
  
<font color="green">// Remove all motors from the Y axis.</font>  
mcAxisUnmapMotors(mInst, Y_AXIS);
```

```
int mcAxisUnregister(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisUnregister(  
    number mInst, number axisId)  
  
int mInst=0;  
  
<font color="green">// Remove AXIS7 from frome the system.</font>  
mcAxisUnregister(mInst, AXIS7);
```

Motion

- [mcMotionClearPlanner](#)
- [mcMotionCyclePlanner](#)
- [mcMotionCyclePlannerEx](#)
- [mcMotionGetAbsPos](#)
- [mcMotionGetAbsPosFract](#)
- [mcMotionGetBacklashAbs](#)
- [mcMotionGetBacklashInc](#)
- [mcMotionGetIncPos](#)
- [mcMotionGetMoveID](#)
- [mcMotionGetPos](#)
- [mcMotionGetProbeParams](#)
- [mcMotionGetRigidTapParams](#)
- [mcMotionGetSyncOutput](#)
- [mcMotionGetThreadParams](#)
- [mcMotionGetThreadingRate](#)
- [mcMotionGetVel](#)
- [mcMotionSetCycleTime](#)
- [mcMotionSetMoveID](#)
- [mcMotionSetPos](#)
- [mcMotionSetProbeComplete](#)
- [mcMotionSetProbePos](#)
- [mcMotionSetStill](#)
- [mcMotionSetThreadingRate](#)
- [mcMotionSetVel](#)
- [mcMotionSync](#)
- [mcMotionThreadComplete](#)

```
int mcMotionClearPlanner(  
    MINSTANCE mInst);  
  
rc = mc.mcMotionClearPlanner(  
    number mInst)  
  
<font color="green">// Clear the planner.</font> MINSTANCE mInst = 0;  
  
int rc = mcMotionClearPlanner(mInst);
```

mcMotionCyclePlanner

C/C++ Syntax:

```
int mcMotionCyclePlanner(MINSTANCE mInst);
```

LUA Syntax:

N/A

Description: Cycle the core planner.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

This function is deprecated in favor of mcMotionCyclePlannerEx().

Usage:

None .

```
int mcMotionCyclePlannerEx(  
    MINSTANCE mInst, execution_t *exInfo);
```

N/A

// Cycle the core planner. MINSTANCE mInst = 0; execution_t exInfo; int rc = mcMotionCyclePlannerEx(mInst, &exInfo); // For more information, see the Sim plugin example.


```
int mcMotionGetAbsPos(
```

```
    MINSTANCE mInst, int motorId, double *val);
```

N/A

None.

```
int mcMotionGetAbsPosFract(  
    MINSTANCE mInst, int motorId, double *val);
```

N/A

None.

```
int mcMotionGetBacklashAbs(  
    MINSTANCE mInst, int motorId, double *pos);
```

N/A

None.

```
int mcMotionGetBacklashInc(  
    MINSTANCE mInst, int motorId, double *pos);
```

N/A

None.

```
int mcMotionGetIncPos(
```

```
    MINSTANCE mInst, int motorId, double *val);
```

N/A

None.

```
int mcMotionGetMoveID(
```

```
    MINSTANCE mInst, int motorId, long *val);
```

N/A

None.

```
int mcMotionGetPos(  
    MINSTANCE mInst, int motorId, double *pos);  
  
pos, rc = mc.mcMotionGetPos(  
    number mInst, number motorId)  
  
<font color="green">// Get the motor 0 position.</font> MINSTANCE  
mInst = 0;  
  
double pos;  
  
int rc = mcMotionGetPos(mInst, 0, &pos);
```

```
int mcMotionGetProbeParams(  
    MINSTANCE mInst, probe_t *probeInfo);
```

N/A

```
<font color="green">// Get the probing parameters from the core.</font>  
MINSTANCE mInst = 0; probe_t pInfo; int  
mcMotionGetProbeParams(mInst, &pInfo);
```

```
int mcMotionGetRigidTapParams(  
    MINSTANCE mInst, tap_t *tapInfo);
```

N/A

```
<font color="green">// Get the tpping parameters from the core.</font>  
MINSTANCE mInst = 0; tap_t tapInfo; int rc =  
mcMotionGetRigidTapParams(mInst, &tapInfo);
```

```
int mcMotionGetSyncOutput(
```

```
    MINSTANCE mInst, int outputQueue, HMCIO *hIo, BOOL *state);
```

N/A

// Retrieve the coordinated outputs/

```
MINSTANCE mInst = 0; execution_t ex;
```

```
mcMotionCyclePlannerEx(mInst, &ex); if (ex.exOutputQueue !=  
EX_NONE) {
```

```
    HMCIO hIo; BOOL state; while (mcMotionGetSyncOutput(m_cid,  
ex.exOutputQueue, &hIo, &state) == MERROR_NOERROR) {
```

// Pair setting the output along with the moves in
(execution_t)ex. }

```
}
```

```
int mcMotionGetThreadParams(  
    MINSTANCE mInst, thread_t *threadInfo);  
  
<font color="green">// </font> MINSTANCE mInst = 0;  
  
thread_t threadInfo;  
  
int mcMotionGetThreadParams(mInst, &threadInfo);
```

```
int mcMotionGetThreadingRate(  
    MINSTANCE mInst, double *ratio);
```

N/A

```
<font color="green">// Get the current threading rate from the core.</font>  
MINSTANCE mInst = 0; double ratio; int rc =  
mcMotionGetThreadingRate(mInst, &ratio);
```

```
int mcMotionGetVel(MINSTANCE mInst, int motorId, double *velocity);  
velocity, rc = mc.mcMotionGetVel(  
    number mInst, number motorId)  
  
<font color="green">// Get the motor velocity for motor 1</font>  
MINSTANCE mInst = 0;  
  
double vel;  
  
int rc = mcMotionGetVel(mInst, 1, &vel);
```

```
int mcMotionSetCycleTime(  
    MINSTANCE mInst, double secs);
```

N/A

```
<font color="green">// Set the cycle time slice.</font> MINSTANCE mInst  
= 0; int mcMotionSetCycleTime(mInst, .001);
```

```
int mcMotionSetMoveID(  
    MINSTANCE mInst, int id);
```

N/A

// Set the currently executing movement ID.
MINSTANCE mInst = 0; int moveId = 10001; //
Should be obtained from the motion controller. int
mcMotionSetMoveID(mInst, moveId);

```
int mcMotionSetPos(
```

```
    MINSTANCE mInst, int motorId double val);
```

N/A

```
<font color="green">// Set the motor position for motor 1.</font>
MINSTANCE mInst = 0; motorCounts = 324255; <font color="green">//
Should be retrieved from the motion controller.</font> int
mcMotionSetPos(mInst, 1, motorCounts);
```

```
int mcMotionSetProbeComplete(
```

```
    MINSTANCE mInst);
```

N/A

```
<font color="green">// Complete a probe operation.</font> MINSTANCE  
mInst = 0; int rc = mcMotionSetProbeComplete(mInst);
```

```
int mcMotionSetProbePos(
```

```
    MINSTANCE mInst, int motorId, double val);
```

N/A

// Set the probed position for motor 0.

```
MINSTANCE mInst = 0; double probedPos = 2314134; <font
```

color="green">// Should be reteived from motion controller latch registers.

```
</font> int rc = mcMotionSetProbePos(mInst, 0, double val);
```

```
int mcMotionSetStill(  
    MINSTANCE mInst, int motorId);
```

N/A

```
<font color="green">// Motor stop report example.</font> MINSTANCE  
mInst = 0; execution_t ex;  
  
mcMotionCyclePlannerEx(m_cid, &ex); switch(ex.exType) {  
  
case EX_STOP_REQ:  
  
    // See if we need to report when the motors are still.  
  
    for (i = 0; i < 8; i++) {  
  
        if (ex.exMotors[i].reportStopped == TRUE) {  
  
            <font color="green">// Report when this motor has completed all  
previous moves!</font> }  
  
    }  
  
    break; ...  
}
```

```
int mcMotionSetThreadingRate(  
    MINSTANCE mInst, double ratio);
```

N/A

```
<font color="green">// Set the threading ratio.</font> MINSTANCE mInst  
= 0; int rc = mcMotionSetThreadingRate(mInst, 1.2);
```

```
int mcMotionSetVel(  
    MINSTANCE mInst, int motorId, double velocity);
```

N/A

```
<font color="green">// Report the velocity for motor 0.</font>  
MINSTANCE mInst = 0; double vel = 2342420;<font color="green">//  
Should be obtained from the motion controller.</font> int  
mcMotionSetVel(mInst, 0, vel);
```

```
int mcMotionSync(
```

```
    MINSTANCE mInst);
```

N/A

```
<font color="green">// Synch core planner positions with the last reported  
motor positions.</font> MINSTANCE mInst = 0; int rc =  
mcMotionSync(mInst);
```

```
int mcMotionThreadComplete(
```

```
    MINSTANCE mInst);
```

N/A

```
<font color="green">// Report that the threading op is complete.</font>
MINSTANCE mInst = 0; int rc = mcMotionThreadComplete(mInst);
```

Operation

- [mcCtlCycleStart](#)
- [mcCtlCycleStop](#)
- [mcCtlDryRunToLine](#)
- [mcCtlEStop](#)
- [mcCtlFeedHold](#)
- [mcCtlFeedHoldState](#)
- [mcCtlGetBlockDelete](#)
- [mcCtlGetEnableFRO](#)
- [mcCtlGetFRO](#)
- [mcCtlGetOptionalStop](#)
- [mcCtlGetRRO](#)
- [mcCtlGetSingleBlock](#)
- [mcCtlGotoZero](#)
- [mcCtlIsInCycle](#)
- [mcCtlReset](#)
- [mcCtlSetBlockDelete](#)
- [mcCtlSetEnableFRO](#)
- [mcCtlSetFRO](#)
- [mcCtlSetOptionalStop](#)
- [mcCtlSetRRO](#)
- [mcCtlSetResetCodes](#)
- [mcCtlSetSingleBlock](#)
- [mcCtlStartMotionDev](#)
- [mcCtlStopMotionDev](#)
- [mcCtlToolChangeManual](#)
- [mcCtlWaitOnCycleStart](#)

```
int mcCtlCycleStart(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlCycleStart(  
    number mInst)  
  
int mInst=0;  
  
<font color="green">// Start controller 0's Gcode file.</font>  
mcCtlCycleStart(mInst);
```

```
int mcCtlCycleStop(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlCycleStop(  
    number mInst)  
  
int mInst=0;  
  
<font color="green">// Stop controller 0's Gcode file.</font>  
mcCtlCycleStop(mInst);
```

```
int mcCtlDryRunToLine(  
    MINSTANCE mInst, int line);  
  
rc = mc.mcCtlDryRunToLine(  
    number mInst, number line)  
  
<font color="green">// Dry run to line 38</font> MINSTANCE mInst = 0;  
  
int rc = mcCtlDryRunToLine(mInst, 38);
```

```
int mcCtlEStop(  
    MINSTANCE mInst);
```

```
rc = mc.mcCtlEStop(  
    number mInst)
```

```
<font color="green">// Put the control in the default state when e-stop is  
asserted.</font> MINSTANCE mInst = 0; int rc = mcCtlEStop(mInst);
```

```
int mcCtlFeedHold(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlFeedHold(  
    number mInst)  
  
int mInst = 0; int rc;  
  
rc = mcCtlFeedHold(mInst); <font color="green">// Pause the motion of  
controller 0.</font> if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Feed hold was successful.</font> }  
}
```

```
int mcCtlFeedHoldState(  
    MINSTANCE mInst, BOOL *InFeedHold);  
  
InFeedHold, rc = mc.mcCtlFeedHoldState(  
    number mInst)  
  
int mInst = 0; int rc;  
  
BOOL fhState; rc = mcCtlFeedHoldState(mInst, &fhState); <font  
color="green">// Pause the motion of controller 0.</font> if (rc ==  
MERROR_NOERROR && fhState == TRUE) {  
  
    <font color="green">// The control is in Feedhold.</font> }  
 
```

```
int mcCntrlGetBlockDelete(  
    MINSTANCE mInst, int deleteId, BOOL *val);  
  
val, rc = mc.mcCntrlGetBlockDelete(  
    number mInst, number deleteId)  
  
<font color="green">// Is block delete 0 on?</font> MINSTANCE mInst =  
0;  
  
BOOL val;  
  
int rc = mcCntrlGetBlockDelete(mInst, 0, &val); if (rc ==  
MERROR_NOERROR) {  
  
    if (val == TRUE) {  
  
        <font color="green">// Is block delete 0 is on!</font> } else {  
  
        <font color="green">// Is block delete 0 is off!</font> }  
  
    }  
}
```

```
int mcCntrlGetEnableFRO(  
    MINSTANCE mInst, BOOL *enable);  
  
eanbled, rc = mc.mcCntrlGetEnableFRO(  
    number mInst)  
  
<font color="green">// Get the current FRO status</font> MINSTANCE  
mInst = 0;  
  
BOOL enabled;  
  
int rc = mcCntrlGetEnableFRO(mInst, &enabled); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> if (enabled == TRUE) {  
  
        <font color="green">// Feed Rate Override is enabled!</font> } else {  
  
        <font color="green">// Feed Rate Override is disabled!</font> }  
  
    }  
}
```

```
int mcCtlGetFRO(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mc.mcCtlGetFRO(  
    number mInst)  
  
<font color="green">// Get the current FRO</font> MINSTANCE mInst =  
0; double fro;  
  
int rc = mcCtlGetFRO(mInst, &fro); if (rc == MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCntrlGetOptionalStop(
    MINSTANCE mInst, BOOL *stop);
stop, rc = mc.mcCntrlGetOptionalStop(
    number mInst)

<font color="green">// See if optional stop is in effect.</font>
MINSTANCE mInst = 0;
BOOL opStop = FALSE;
int rc = mcCntrlGetOptionalStop(mInst, &opStop); if (rc ==
MERROR_NOERROR) {

    if (opStop == TRUE) {

        <font color="green">// Optional Stop is in effect!
    } else {

        <font color="green">// Optional Stop is not in effect!
    }
}

</font></font>
```

```
int mcCtlGetRRO(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mc.mcCtlGetRRO(  
    number mInst)  
  
<font color="green">// Get the current RRO</font> MINSTANCE mInst =  
0; double rro;  
  
int rc = mcCtlGetRRO(mInst, &rro); if (rc == MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCntrlGetSingleBlock(  
    MINSTANCE mInst, BOOL *sbState);  
  
sbState, rc = mc.mcCntrlGetSingleBlock(  
    number mInst)  
  
<font color="green">// Get the state of single block.</font> int mInst=0;  
BOOL IsOn = FALSE; mcCntrlGetSingleBlock(mInst, &IsOn);
```

```
int mcCtlGotoZero(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlGotoZero(  
    number mInst)  
  
int mInst = 0;  
  
mcCtlGotoZero(mInst); <font color="green">// Move controller instance 0  
to x,y,z,a,b,c zero.</font>
```

```
int mcCtlIsInCycle(  
    MINSTANCE mInst, BOOL *cycle);  
  
cycle, rc = mc.mcCtlIsInCycle(  
    number mInst)  
  
int mInst = 0;  
  
BOOL InCycle = FALSE; bool RunningFile = false;  
  
int rc = mcCtlIsInCycle(mInst, &InCycle); <font color="green">// See if a  
G code file is running.</font> if(rc == MERROR_NOERROR && InCycle  
== TRUE){  
  
    RunningFile = true; }
```

```
int mcCtlReset(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlReset(  
    number mInst)  
  
MINSTANCE mInst = 0; mcCtlReset(mInst); <font color="green">//  
Reset controller instance 0.</font>
```

```
int mcCtlSetBlockDelete(  
    MINSTANCE mInst, int deleteID, BOOL enabled);  
  
rc = mc.mcCtlSetBlockDelete(  
    number mInst, number deleteID, number enabled)  
  
MINSTANCE mInst = 0;  
  
<font color="green">// Enable block delete level 0.</font>  
mcCtlSetBlockDelete(mInst, 0, TRUE);
```

```
int mcCtlSetEnableFRO(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcCtlSetEnableFRO(  
    number mInst, number enable);  
  
<font color="green">// Enable feed rate override</font> MINSTANCE  
mInst = 0;  
  
int rc = mcCtlSetEnableFRO(mInst, TRUE);
```

```
int mcCtlSetFRO(  
    MINSTANCE mInst, double percent);  
  
rc = mc.mcCtlSetFRO(  
    number mInst, number percent);  
  
<font color="green">// Set feed rate override to 150%</font>  
MINSTANCE mInst = 0; int rc = mcCtlSetFRO(mInst, 150.0);
```

```
int mcCtlSetOptionalStop(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcCtlSetOptionalStop(  
    number mInst number enable)  
  
<font color="green">// Enable optional stop.</font> MINSTANCE mInst =  
0;  
  
int rc = mcCtlSetOptionalStop(mInst, TRUE);
```

```
int mcCtlSetRRO(
```

```
    MINSTANCE mInst, double percent);
```

```
rc = mc.mcCtlSetRRO(
```

```
    number mInst, number percent);
```

```
<font color="green">// Set feed rapid override to 50%</font>
MINSTANCE mInst = 0; int rc = mcCtlSetRRO(mInst, 50.0);
```

```
int mcCtlSetResetCodes(  
    MINSTANCE mInst, const char *resetCodes);  
  
rc = mc.mcCtlSetResetCodes(  
    number mInst, string resetCodes)  
  
<font color="green">// Set the G code used to reset the controller instance  
0.</font> MINSTANCE mInst = 0;  
  
int rc = mcCtlSetResetCodes(mInst, "G40 G20 G90 G52 X0 Y0  
Z0\nG92.1 G69");
```

```
int mcCtlSetSingleBlock(  
    MINSTANCE mInst, BOOL enable);  
  
<font color="green">// Enable single block mode.</font> MINSTANCE  
mInst = 0;  
  
mcCtlSetSingleBlock(mInst, TRUE);
```

```
int mcCtlStartMotionDev(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlStartMotionDev(  
    number mInst)  
  
<font color="green">// Start the selected motion device.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcCtlStartMotionDev(mInst);
```

```
int mcCtlStopMotionDev(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlStopMotionDev(  
    number mInst);  
  
<font color="green">// Stop the selected motion device.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcCtlStopMotionDev(mInst);
```

```
int mcCtlToolChangeManual(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlToolChangeManual(  
    number mInst)
```

```
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance(); local rc = mc.mcCtlToolChangeManual(inst);
```

```
int mcCtlWaitOnCycleStart(  
    MINSTANCE mInst, const char *msg, int timeOutMs);  
  
rc = mc.mcCtlWaitOnCycleStart(  
    number mInst, string msg, number timeOutMs);  
  
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance();  
  
local rc = mc.mcCtlWaitOnCycleStart(inst, "Please press Cycle Start.",  
1000);
```

GUI

- [mcCtlCleanup](#)
- [mcCtlInit](#)
- [mcFixtureLoadFile](#)
- [mcFixtureSaveFile](#)
- [mcGuiGetWindowHandle](#)
- [mcGuiSetFocus](#)
- [mcGuiSetWindowHandle](#)
- [mcToolPathCreate](#)
- [mcToolPathDelete](#)
- [mcToolPathGenerate](#)
- [mcToolPathGenerateAbort](#)
- [mcToolPathGeneratedPercent](#)
- [mcToolPathGetAAxisPosition](#)
- [mcToolPathGetARotationAxis](#)
- [mcToolPathGetAxisColor](#)
- [mcToolPathGetBackColor](#)
- [mcToolPathGetDrawLimits](#)
- [mcToolPathGetExecution](#)
- [mcToolPathGetFollowMode](#)
- [mcToolPathGetGenerating](#)
- [mcToolPathGetLeftMouseDn](#)
- [mcToolPathGetLeftMouseUp](#)
- [mcToolPathGetPathColor](#)
- [mcToolPathIsSignalMouseClicks](#)
- [mcToolPathSetAAxisPosition](#)
- [mcToolPathSetARotationAxis](#)
- [mcToolPathSetAxisColor](#)
- [mcToolPathSetBackColor](#)
- [mcToolPathSetDrawLimits](#)
- [mcToolPathSetFollowMode](#)

- [mcToolPathSetColor](#)
- [mcToolPathSetSignalMouseClicks](#)
- [mcToolPathSetView](#)
- [mcToolPathSetZoom](#)
- [mcToolPathUpdate](#)

```
int mcCtlCleanup(MINSTANCE mInst);
```

N/A

None.

```
<font color="green">// Cleanup core instance 0</font> MINSTANCE  
mInst = 0; int rc = mcCtlCleanup(mInst);
```

```
MINSTANCE mcCtlInit(
```

```
    const char *ProfileName, int ControllerID);
```

N/A

```
MINSTANCE mInst; mInst = mcCtlInit("Mach4Mill", mInst); if (mInst  
>= 0) {
```

```
    <font color="green">// A valid instance has been initialized!</font> }
```

```
int mcFixtureLoadFile(  
    MINSTANCE mInst, const char *FileToLoad);  
  
rc = mc.mcFixtureLoadFile(  
    number mInst, string FileToLoad);  
  
<font color="green">// Load a fixture table file.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcFixtureLoadFile(mInst, "MyFixtureTable.dat");
```

```
int mcFixtureSaveFile(  
    MINSTANCE mInst);  
  
rc = mc.mcFixtureSaveFile(  
    number mInst),  
  
<font color="green">// Save the fixture table file.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcFixtureSaveFile(mInst);
```

```
int mcGuiGetWindowHandle(  
    MINSTANCE mInst, void **handle);
```

N/A

```
<font color="green">// </font> MINSTANCE mInst = 0; void *guiHandle;  
rc = mcGuiGetWindowHandle(mInst, &guiHandle);
```

```
int mcGuiSetCallback(  
    MINSTANCE mInst, void *fp);
```

N/A

```
<font color="green"> /* In the GUI code */ </font> MCP_API <font  
color="blue">int</font> MCP_APIENTRY mcGUIMsg(MINSTANCE  
mInst, <font color="blue">long</font> msg, <font  
color="blue">long</font> wparam, <font color="blue">long</font>  
lparam) {  
  
    <font color="green"> // Process messages...</font>  
    return(MERROR_NOERROR); }
```

```
MINSTANCE mInst = 0; mcGuiSetCallback(mInst, &mcGUIMsg);
```

```
int mcGuiSetFocus(  
    MINSTANCE mInst, BOOL focus);
```

N/A

// Set focus to the GUI's main window
MINSTANCE mInst = 0; int rc = mcGuiSetFocus(mInst, TRUE);

```
int mcGuiSetWindowHandle(  
    MINSTANCE mInst, void *handle);
```

N/A

```
<font color="green">// Set the GUI main window handle.</font>  
MINSTANCE mInst = 0; mcGuiSetWindowHandle(0, this);
```

```
int mcToolPathCreate(  
    MINSTANCE mInst, void *window);
```

N/A

```
wxPanel* m_tpTP;
```

```
MINSTANCE mInst = 0; m_tpTP = new wxPanel(itemPanel129,  
ID_TOOLPATH_PANEL, wxDefaultPosition, wxDefaultSize,  
wxTAB_TRAVERSAL ); rc = m_tpTP->Show(); <font color="green">//  
Pass the handle of the tool path.</font> mcToolPathCreate(mInst, m_tpTP-  
>GetHWND());
```

```
int mcToolPathDelete(  
    MINSTANCE mInst, void *parent);
```

N/A

```
<font color="green">// Delete an existing tool path.</font> MINSTANCE  
mInst = 0; HWND parent; <font color="green">// A previously valid  
window handle.</font> int rc = mcToolPathDelete(mInst, parent);
```

```
int mcToolPathGenerate(  
    MINSTANCE mInst);  
  
rc = mc.mcToolPathGenerate(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
<font color="green">// Regenerate the toolpaths for controller instance  
0</font> int rc = mcToolPathGenerate(mInst);
```

```
int mcToolPathGenerateAbort(  
    MINSTANCE mInst);  
  
rc = mc.mcToolPathGenerateAbort(  
    number mInst);  
  
MINSTANCE mInst = 0;  
  
<font color="green">// Abort the regeneration of the toolpath for controller  
0.</font> int rc = mcToolPathGenerateAbort(mInst);
```

```
int mcToolPathGeneratedPercent(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mc.mcToolPathGeneratedPercent(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
double pdone = 0;  
  
  
  
while (pdone != 100) {  
  
    <font color="green">// Get the percent of the file that is loaded.</font>  
    mcToolPathGeneratedPercent(mInst, &pdone); <font color="green">// Post  
    the pdone to some dialog.</font> Sleep(250); }
```

```
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double  
*yPos, double *zPos);
```

```
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double  
*yPos, double *zPos);
```

```
<font color="green">// </font> MINSTANCE mInst = 0;  
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double  
*yPos, double *zPos);
```

mcToolPathGetARotationAxis

C/C++ Syntax:

```
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

LUA Syntax:

```
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

Usage:

```
//  
MINSTANCE mInst = 0;  
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

```
int mcToolPathGetAxisColor(  
    MINSTANCE mInst, unsigned long *axiscolor, unsigned long  
    *limitcolor);  
  
axiscolor, limitcolor, rc = mc.mcToolPathGetAxisColor(  
    number mInst)  
  
<font color="green">// Get the axis and limit colors.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcToolPathGetAxisColor(MINSTANCE mInst, unsigned long  
    *axiscolor, unsigned long *limitcolor);
```

```
int mcToolPathGetBackColor(
    MINSTANCE mInst, unsigned long *topcolor, unsigned long *botcolor);
topcolor, botcolor, rc = mc.mcToolPathGetBackColor(
    number mInst)
MINSTANCE mInst = 0;
unsigned long topcol, botcol;

<font color="green">// Get the tool background top and bottom
colors</font> int rc = mcToolPathGetBackColor(m_inst, &topcol,
&botcol);
```

```
int mcToolPathGetDrawLimits(  
    MINSTANCE mInst, BOOL *drawlimits);  
  
drawlimits, rc = mcToolPathGetDrawLimits(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
BOOL drawlimits = FALSE;  
  
<font color="green">// See if the soft limits bounding box is beeing shown  
in the toolpath.</font> int rc = mcToolPathGetDrawLimits(mInst,  
&drawlimits);
```

```
int mcToolPathGetExecution(
```

```
    MINSTANCE mInst, unsigned long exNum, void **data, unsigned long
    *len);
```

N/A

```
struct Executions

{
    bool linear : 1; //Is this a linear move?

    bool rapid : 1; //Is it a Rapid move?

    bool inc : 1; //Is this an inc move?

    bool comp : 1; bool rotation : 1; //arc direction
    bool UsedAxis0 : 1; bool UsedAxis1 : 1;
    bool UsedAxis2 : 1; bool UsedAxis3 : 1;
    bool UsedAxis4 : 1; bool UsedAxis5 : 1;
    bool UsedAxis6 : 1; bool UsedAxis7 : 1;
    bool UsedAxis8 : 1; bool UsedAxis9 : 1;
    bool UsedAxis10 : 1; float end[6];
    float center[3]; float normal[3];
    unsigned int LineNumber; // Line number in the
    file...

    unsigned int ExecutionID; // Number of the move from the Gcode
    interperter
    float FeedRate; // Feedrate of the move
    unsigned char ToolNumber; // Tool number of the move used to show offsets..

    unsigned char Fixture; // The fixture.

};

<font color="green">// </font> MINSTANCE mInst = 0;

unsigned long exNum = 0;

unsigned long len;

void *data;

int rc;

struct Executions *ex;
```

```
<font color="green">// Get the length of the move execution structure first.  
</font> while (mcToolPathGetExecution(mInst, exNum, NULL, &len) ==  
MERROR_NOERROR) {  
  
    <font color="green">// Allocate some mem to receive the data.</font>  
    data = malloc(len); if (data != NULL) {  
  
        <font color="green">// Get the data.</font> if  
(mcToolPathGetExecution(mInst, exNum, &data, &len) ==  
MERROR_NOERROR) {  
  
        <font color="green">// cast the void pointer to the current structure.  
</font> ex = (struct Executions *)data; ...  
  
        free(data); }  
  
    }  
}
```

```
int mcToolPathGetFollowMode(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mc.mcToolPathGetFollowMode(  
    number mInst)  
  
<font color="green">// See if jog follow is enabled.</font> MINSTANCE  
mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcToolPathGetFollowMode(mInst, &enabled);
```

```
int mcToolPathGetGenerating(  
    MINSTANCE mInst, int *pathGenerating);  
  
pathGenerating, rc = mc.mcToolPathGetGenerating(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
BOOL IsGen = FALSE;  
  
<font color="green">// See if the tool path is generating.</font> int rc =  
mcGetPathGenerating(mInst, &IsGen);
```

```
int mcToolPathGetLeftMouseDn(  
    MINSTANCE mInst, double *x, double *y, double *z);  
  
x, y, z, rc = mc.mcToolPathGetLeftMouseDn(  
    number mInst)  
  
<font color="green">// Get the mouse coordinates in the tool path.</font>  
MINSTANCE mInst = 0;  
  
double x, y, z;  
  
int rc = mcToolPathGetLeftMouseDn(MINSTANCE mInst, double *x,  
    double *y, double *z);
```

```
int mcToolPathGetLeftMouseUp(  
    MINSTANCE mInst, double *x, double *y, double *z);  
  
x, y, z, rc = mc.mcToolPathGetLeftMouseUp(  
    number mInst)  
  
<font color="green">// Get the mouse coordinates in the tool path.</font>  
MINSTANCE mInst = 0;  
  
double x, y, z;  
  
int rc = mcToolPathGetLeftMouseUp(MINSTANCE mInst, double *x,  
    double *y, double *z);
```

```
int mcToolPathGetPathColor(  
    MINSTANCE mInst, unsigned long *rapidcolor, unsigned long  
    *linecolor, unsigned long *arccolor, unsigned long *highlightcolor);  
  
rapidcolor, linecolor, arccolor, highlightcolor, rc =  
mcToolPathGetPathColor(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
unsigned long rapcol, lincol, arccol, highcol; <font color="green">// Get the  
tool path colors</font> int rc = mcToolPathGetPathColor(m_inst, &rapcol,  
&lincol, &arccol, &highcol);
```

```
int mcToolPathIsSignalMouseClicks(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mcToolPathIsSignalMouseClicks(  
    number mInst)  
  
<font color="green">// See if we are signaling mouse click events in the  
tool path.</font> MINSTANCE mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcToolPathIsSignalMouseClicks(mInst, &enabled);
```

```
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double  
yPos, double zPos);
```

```
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double  
yPos, double zPos);
```

```
<font color="green">// </font> MINSTANCE mInst = 0;  
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double  
yPos, double zPos);
```

mcToolPathSetARotationAxis

C/C++ Syntax:

```
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

LUA Syntax:

```
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

Usage:

```
//  
MINSTANCE mInst = 0;  
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

```
int mcToolPathSetAxisColor(
    MINSTANCE mInst, unsigned long axiscolor, unsigned long limitcolor);

rc = mc.mcToolPathSetAxisColor(
    number mInst, number axiscolor, number limitcolor)

MINSTANCE mInst = 0;
unsigned long axiscolor;
unsigned long limitcolor;

//red

axiscolor = (255<<0);//red axiscolor += (0<<8);//green axiscolor +=
(0<<16);//blue

//Yellow

limitcolor = (255<<0);//red limitcolor += (255<<8);//green limitcolor +=
(128<<16);//blue

<font color="green">// Set the axis and limit colors</font> int rc =
mcToolPathSetAxisColor(mInst, axiscolor, limitcolor);
```

```
int mcToolPathSetBackColor(
    MINSTANCE mInst, unsigned long topcolor, unsigned long botcolor);
rc = mc.mcToolPathSetBackColor(
    number mInst, number topcolor, number botcolor)
MINSTANCE mInst = 0;
unsigned long topcolor;
```

```
unsigned long botcolor
```

```
//Dark Blue Top
```

```
topcolor = (28<<0); topcolor += (28<<8); topcolor += (213<<16); //Light  
Blue Bottom
```

```
botcolor = (164<<0); botcolor += (156<<8); botcolor += (228<<16);
```

```
<font color="green">// Set the background color</font> int rc =  
mcToolPathSetBackColor(mInst, topcolor, botcolor);
```

```
int mcToolPathSetDrawLimits(  
    MINSTANCE mInst, BOOL drawlimits);  
  
rc = mc.mcToolPathSetDrawLimits(  
    number mInst, number drawlimits)  
  
<font color="green">// Turn on soft limits in the tool path.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcToolPathSetDrawLimits(mInst, TRUE);
```

```
int mcToolPathSetFollowMode(  
    MINSTANCE mInst, BOOL enabled);  
  
rc = mc.mcToolPathSetFollowMode(  
    number mInst, number enabled)  
  
<font color="green">// Turn on tool path jog following.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcToolPathSetFollowMode(mInst, TRUE);
```

```
int mcToolPathSetPathColor(  
    MINSTANCE mInst, unsigned long rapidcolor, unsigned long linecolor,  
    unsigned long arccolor, unsigned long highlightcolor);  
  
rc = mc.mcToolPathSetPathColor(  
    number mInst, number rapidcolor, number linecolor, number arccolor,  
    number highlightcolor)  
  
<font color="green">// Set the tool path colors</font> MINSTANCE mInst  
= 0;  
  
unsigned long rapidcolor;  
  
unsigned long linecolor;  
  
unsigned long arccolor;  
  
unsigned long highlightcolor;  
  
  
  
rapidcolor = (254<<0);//Red rapidcolor += (0<<8);//Green rapidcolor +=  
(0<<16);//Blue  
  
linecolor = (0<<0);//Red  
  
linecolor += (254<<8);//Green linecolor += (0<<16);//Blue  
  
arccolor = (0<<0);//Red  
  
arccolor += (254<<8);//Green arccolor += (0<<16);//Blue  
  
highlightcolor = (254<<0);//Red highlightcolor += (254<<8);//Green  
highlightcolor += (254<<16);//Blue
```

```
int rc = mcToolPathSetPathColor(mInst, rapidcolor, linecolor, arccolor,  
highlightcolor);
```

```
int mcToolPathSetSignalMouseClicks(  
    MINSTANCE mInst, BOOL enabled);  
  
rc = mc.mcToolPathSetSignalMouseClicks(  
    number mInst, number enabled)  
  
<font color="green">// Turn on mouse click events in the tool path.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcToolPathSetSignalMouseClicks(mInst, TRUE);
```

```
int mcToolPathSetView(
```

```
    MINSTANCE mInst, void *parent, int view);
```

N/A

```
<font color="green">// Set the tool path orientation to the ISO view.</font>
MINSTANCE mInst = 0;
```

```
HWND parent; <font color="green">// A valid window handle.</font> int
rc = mcToolPathSetView(mInst, parent, MCTPVIEW_ISO);
```

```
int mcToolPathSetZoom(
```

```
    MINSTANCE mInst, void *parent, double zoom);
```

N/A

```
<font color="green">// Set the tool path zoom percentage to 150%</font>
MINSTANCE mInst = 0;
```

```
HWND parent; <font color="green">// A valid window handle.</font> int
rc = mcToolPathSetZoom(mInst, parent, 1.5);
```

```
int mcToolPathUpdate(  
    MINSTANCE mInst, void *parent);
```

N/A

```
MINSTANCE mInst = 0; HWND parent; <font color="green">// A valid  
window handle.</font> int mcToolPathUpdate(mInst, parent);
```

Profile (INI) Settings

- [mcProfileDeleteKey](#)
- [mcProfileDeleteSection](#)
- [mcProfileExists](#)
- [mcProfileFlush](#)
- [mcProfileGetDouble](#)
- [mcProfileGetInt](#)
- [mcProfileGetName](#)
- [mcProfileGetString](#)
- [mcProfileReload](#)
- [mcProfileSave](#)
- [mcProfileWriteDouble](#)
- [mcProfileWriteInt](#)
- [mcProfileWriteString](#)

```
int mcProfileDeleteKey(  
    MINSTANCE mInst, const char *section, const char *key);  
  
rc = mc.mcProfileDeleteKey(  
    number mInst, string section, string key)  
  
MINSTANCE mInst = 0;  
  
mcProfileDeleteKey(mInst , "P_Port", "Frequency");
```

```
int mcProfileDeleteSection(  
    MINSTANCE mInst, const char *section);  
  
rc = mc.mcProfileDeleteSection(  
    number mInst, string section)  
  
MINSTANCE mInst = 0;  
  
mcProfileDeleteKey(mInst , "P_Port");
```

```
int mcProfileExists(  
    MINSTANCE mInst, const char *section, const char *key);  
  
rc = mc.mcProfileExists(  
    number mInst, string section, string key)  
  
MINSTANCE mInst = 0;  
  
mcProfileWriteInt(mInst, "P_Port", "Frequency");
```

```
int mcProfileFlush(
```

```
    INSTANCE mInst);
```

```
rc = mc.mcProfileFlush(
```

```
    number mInst)
```

```
MINSTANCE mInst = 0; mcProfileFlush(mInst); <font color="green">//  
Flush changes to the INI file.</font>
```

```
mcProfileGetDouble(
```

```
    MINSTANCE mInst, const char *section, const char *key, double  
    *retval, double defval);
```

```
retval, rc = mc.mcProfileGetDouble(
```

```
    number mInst, string section, string key, number defval)
```

```
double rval=0;
```

```
MINSTANCE mInst = 0;
```

```
mcProfileGetDouble(mInst , "P_Port", "Frequency", &rval, 25.34);
```

```
int mcProfileGetInt(  
    MINSTANCE mInst, const char *section, const char *key, long *retval,  
    long defval);  
  
retval, rc = mc.mcProfileGetInt(  
    number mInst, string section, string key, number defval)  
  
long rval=0;  
  
int mInst=0;  
  
mcProfileGetInt(mInst , "P_Port", "Frequency", &rval, 25000);
```

```
int mcProfileGetName(  
    MINSTANCE mInst, char *buff, size_t bufsize);  
  
name, rc = mc.mcProfileGetName(  
    number mInst)  
  
MINSTANCE mInst = 0; char buff[80];  
  
memset(buff, 0, 80); mcProfileGetName(mInst, buff, 80);
```

```
int mcProfileGetString(  
    MINSTANCE mInst, const char *section, const char *key, char *buff,  
    long bufsize, const char *defval);  
  
buff, rc = mc.mcProfileGetString(  
    MINSTANCE mInst, string section, string key, string defval);  
  
MINSTANCE m_inst = 0;  
  
char *key = "BufferedTime"; char buff[80];  
  
memset(buff, 0, 80);  
  
mcProfileGetString(mInst , "SomeSection", key, buff, 80, "0.100");
```

```
int mcProfileReload(  
    INSTANCE mInst);  
  
rc = mc.mcProfileReload(  
    number mInst)  
  
MINSTANCE mInst = 0; mcProfileReload(mInst); <font color="green">//  
Reload the settings from the INI file.</font>
```

```
int mcProfileSave(  
    INSTANCE mInst);  
  
rc = mc.mcProfileSave(  
    number mInst)
```

```
MINSTANCE mInst = 0; mcProfileSave(mInst); <font color="green">//  
Flush the settings to the INI file.</font>
```

```
int mcProfileWriteDouble(  
    MINSTANCE mInst, const char *section, const char *key, double val);  
  
rc = mc.mcProfileWriteDouble(  
    number mInst, string section, string key, double val)  
  
MINSTANCE mInst=0;  
  
mcProfileWriteDouble(m_cid , "P_Port", "Frequency", 45000);
```

```
int mcProfileWriteInt(  
    MINSTANCE mInst, const char *section, const char *key, long val);  
  
int mcProfileWriteInt(  
    number mInst, string section, string key, number val)  
  
MINSTANCE mInst = 0;  
  
mcProfileWriteInt(m_cid , "P_Port", "Frequency", 45000);
```

```
int mcProfileWriteString(  
    MINSTANCE mInst, const char *section, const char *key, const char  
    *val);  
  
rc = mc.mcProfileWriteString(  
    number mInst, string section, string key, string val)  
  
MINSTANCE m_inst = 0;  
  
char *key = "BufferedTime"  
  
double BuffTime = .250;  
  
char val[80];  
  
sprintf(val, "%.4f", BuffTime); mcProfileWriteString(mInst, "Darwin", key,  
val);
```



Home



Up a level



Previous



Next

General

- [mcCtlConfigStart](#)
- [mcCtlConfigStop](#)
- [mcCtlCreateLocalVars](#)
- [mcCtlEnable](#)
- [mcCtlGcodeExecute](#)
- [mcCtlGcodeExecuteWait](#)
- [mcCtlGcodeInterpGetData](#)
- [mcCtlGcodeInterpGetPos](#)
- [mcCtlGetBuild](#)
- [mcCtlGetComputerID](#)
- [mcCtlGetCoolantDelay](#)
- [mcCtlGetCwd](#)
- [mcCtlGetDiaMode](#)
- [mcCtlGetDistToGo](#)
- [mcCtlGetInstanceHandle](#)
- [mcCtlGetLocalComment](#)

- [mcCtlGetLocalVar](#)
- [mcCtlGetLocalVarFlag](#)
- [mcCtlGetLogging](#)
- [mcCtlGetMachDir](#)
- [mcCtlGetMistDelay](#)
- [mcCtlGetModalGroup](#)
- [mcCtlGetMode](#)
- [mcCtlGetOffset](#)
- [mcCtlGetPoundVar](#)
- [mcCtlGetRegister](#)
- [mcCtlGetRunTime](#)
- [mcCtlGetSpindleSpeed](#)
- [mcCtlGetState](#)
- [mcCtlGetStateName](#)
- [mcCtlGetStats](#)
- [mcCtlGetToolOffset](#)
- [mcCtlGetUnitsCurrent](#)
- [mcCtlGetUnitsDefault](#)
- [mcCtlGetValue](#)
- [mcCtlGetVersion](#)

- [mcCtlIsStill](#)
- [mcCtlMachineStateClear](#)
- [mcCtlMachineStatePop](#)
- [mcCtlMachineStatePush](#)
- [mcCtlMacroAlarm](#)
- [mcCtlMacroStop](#)
- [mcCtlMdiExecute](#)
- [mcCtlPluginConfig](#)
- [mcCtlPluginDiag](#)
- [mcCtlProbeFileClose](#)
- [mcCtlProbeFileOpen](#)
- [mcCtlProbeGetStrikeStatus](#)
- [mcCtlSetCoolantDelay](#)
- [mcCtlSetDiaMode](#)
- [mcCtlSetMistDelay](#)
- [mcCtlSetMode](#)
- [mcCtlSetPoundVar](#)
- [mcCtlSetStats](#)
- [mcCtlSetValue](#)
- [mcFileHoldAquire](#)

- [mcFileHoldReason](#)
- [mcFileHoldRelease](#)

```
int mcCtlConfigStart(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlConfigStart(  
    number mInst);  
  
<font color="green">// Enter the configure state</font> MINSTANCE  
mInst = 0;  
  
if (mcCtlConfigStart(mInst) == MERROR_NOERROR) {  
  
    <font color="green">// Successfully entered the configure state!</font> }
```

```
int mcCtlConfigStop(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlConfigStop(  
    number mInst);  
  
<font color="green">// Leave the configure state.</font> MINSTANCE  
mInst = 0; int rc = mcCtlConfigStop(mInst); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Successfully exited the configure state.</font> }
```

```
int mcCtlCreateLocalVars(  
    MINSTANCE mInst,  
    const char *lineParams, unsigned long *handle)  
hParams, rc = mc.mcCtlCreateLocalVars(  
    number mInst,  
    string lineParams);  
  
<font color="green">--Example m19 script that takes parameters</font>  
<br/> function m19(hParam) <font color="green">--hParam is a handle to  
all parameters on same line as m19 --The R and P params are optional. M19  
R90 P0 for example.</font> <font color="green">--R is angle from 0 to  
360.</font> <font color="green">--P is direction: 0 == shortest angle, 1 ==  
clockwise, 2 == counterclockwise</font> <font color="green">--Fanuc  
uses S for angle</font> local inst = mc.mcGetInstance() local pcallRet =  
true
```

local stat = 1

local msgPre = "M19 macro says "

local msg = "M19 Completed successfully"

local varR = 0

local varP = 0

if (hParam ~= nil) then local rc, flagR, flagP

```
    flagR, rc = mc.mcCtlGetLocalVarFlag(inst, hParam, mc.SV_R) flagP, rc  
= mc.mcCtlGetLocalVarFlag(inst, hParam, mc.SV_P) if (flagR == 1) then  
<font color="green">--Check that the flag has been set so we do not get an  
unexpected value for mc.SV_R</font> varR = mc.mcCtlGetLocalVar(inst,  
hParam, mc.SV_R) if (varR < 0) or (varR > 360) then <font
```

```
color="green">--It is out of range</font> msg = string.format("varR == " ..  
varR .. " and is out of range (0-360)") mc.mcCtlCycleStop(inst) return
```

```
end
```

```
end
```

```
if (flagP == 1) then <font color="green">--Check that the flag has been  
set so we do not get an unexpected value for mc.SV_P</font> varP =  
mc.mcCtlGetLocalVar(inst, hParam, mc.SV_P) <font color="green">--  
fixup the values to pass to the spindleorient script.</font> if (varP == 0)  
then
```

```
varP = mc.MC_SPINDLE_STOP <font color="green">-- 0</font> elseif  
(varP == 1) then varP = mc.MC_SPINDLE_FWD <font color="green">--  
1</font> elseif (varP == 2) then varP = mc.MC_SPINDLE_REV <font  
color="green">-- -1</font> else
```

```
msg = string.format("varP == " .. varP .. " and is out of range (0-2)")  
mc.mcCtlCycleStop(inst) return
```

```
end
```

```
end
```

```
end
```

```
pcallRet, stat, msg = pcall(spindleorient, varR, varP); <font  
color="green">--Call the spindleorient.mcs script</font>  
mc.mcCtlSetLastError(inst, msgPre .. msg)
```

```
if (pcallRet == false) then mc.mcCtlCycleStop(inst) end
```

end

if (mc.mcInEditor() == 1) then

 --We are testing the script in the editor.
 --Fab up some paramters to pass to our m19()
 function. local inst = mc.mcGetInstance() local hParams, rc = <font
 color="blue">mc.mcCtlCreateLocalVars(inst, "R320 P0")<font
 color="green">--Like we are testing "M19 R320 P0".
 m19(hParams) --Call m19() with hParams. end

```
int mcCtlEnable(  
    MINSTANCE mInst, BOOL state);  
  
rc = mc.mcCtlEnable(  
    number mInst, number state)  
  
<font color="green">// Enable the control.</font> MINSTANCE mInst = 0;  
int rc = mcCtlEnable(mInst, TRUE);
```

```
int mcCtlGcodeExecute(
    MINSTANCE mInst, const char *commands);

rc = mc.mcCtlGcodeExecute(
    number mInst, string commands)

<font color="green">// Execute spindle stop and rapid to 0, 0.</font>
MINSTANCE mInst = 0;

int rc;

rc = mcCtlGcodeExecute(mInst, "M05\nG00 X0 Y0"); if (rc ==
MERROR_NOERROR) {

    <font color="green">// The G code was submitted for processing.</font>
    <font color="green">// However, the function return immediately and does
    not wait on the G code to finish.</font> }
```

```
int mcCtlGcodeExecuteWait(  
    MINSTANCE mInst, const char *commands);  
  
rc = mcCtlGcodeExecuteWait(  
    number mInst, string commands)  
  
<font color="green">// Execute spindle stop and rapid to 0, 0.</font>  
MINSTANCE mInst = 0;  
  
int rc;  
  
rc = mcCtlGcodeExecuteWait(mInst, "M05\nG00 X0 Y0"); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// The G code was submitted for processing and has  
    completed.</font> }  
 
```

```
int mcCtlGcodeInterpGetData(
```

```
    MINSTANCE mInst, interperter_t *data);
```

N/A

```
struct Interperter_info {
```

```
    double ModalGroups[MC_MAX_GROUPS]; double FeedRate; double
    SpindleSpeed; int SpindleDirection; BOOL Mist; BOOL Flood; int
    ToolNumber; int HeightRegister; int DiaRegister; };
```

```
typedef struct Interperter_info interperter_t;
```

```
<font color="green">// Get the interpreter information.</font>
MINSTANCE mInst = 0;
```

```
interperter_t data;
```

```
int rc;
```

```
rc = mcCtlGcodeInterpGetData(mInst, &data); if (rc ==
MERROR_NOERROR) {
```

```
<font color="green">// Success!</font> }
```

```
int mcCtlGcodeInterpGetPos(  
    MINSTANCE mInst, int axisId, double *pos);  
  
pos, rc = mc.mcCtlGcodeInterpGetPos(  
    number mInst, number axisId)  
  
<font color="green">// Get the current buffered X axis position.</font>  
MINSTANCE mInst = 0; double pos;  
  
int rc;  
  
rc = int mcCtlGcodeInterpGetPos(mInst, X_AXIS, &pos); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetBuild(  
    MINSTANCE mInst, char *buf, size_t bufsize);  
  
build, rc = mc.mcCtlGetBuild(  
    number mInst)  
  
<font color="green">// Get the core's build number</font> MINSTANCE  
mInst = 0; char buildBuf[80];  
  
int rc = mcCtlGetBuild(mInst, buildBuf, sizeof(buildBuf)); if (rc ==  
MERROR_NOERROR) {  
  
    printf("The current build is %s.n", buildBuf); }
```

```
int mcCntrlGetComputerID(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCntrlGetComputerID(  
    number mInst)  
  
<font color="green">// Get the ID(s) for this host</font> MINSTANCE  
mInst = 0;  
  
int rc = mcCntrlGetComputerID(mInst, buf, sizeof(buf)); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Parse buf to get the ID(s)</font> }  
 
```

```
int mcCtlGetCoolantDelay(
    MINSTANCE mInst, double *secs);

sec, rc = mc.mcCtlGetCoolantDelay(
    number mInst)

<font color="green">// Get the coolant delay.</font> MINSTANCE mInst =
0;

double secs;

int rc = mcCtlGetCoolantDelay(mInst, &secs); if (rc ==
MERROR_NOERROR) {

<font color="green">// Success!</font> }
```

```
int mcCtlGetCwd(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCtlGetCwd(  
    number mInst)  
  
<font color="green">// Get current working directory.</font>  
MINSTANCE mInst = 0;  
  
char curDir[255];  
  
int rc = mcCtlGetCwd(mInst, curDir, sizeof(curDir)); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> }  
 
```

```
int mcCtlGetDiaMode(  
    MINSTANCE mInst, BOOL *dia);  
  
dia, rc = mc.mcCtlGetDiaMode(  
    number mInst)  
  
<font color="green">// Get the lathe diameter mode.</font> MINSTANCE  
mInst = 0; BOOL Dia;  
  
int rc = mcCtlGetDiaMode(mInst, &Dia); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetDistToGo(  
    MINSTANCE mInst, int axisId, double *togo);  
  
togo, rc = int mcCtlGetDistToGo(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double togo=0;  
  
mcCtlGetDistToGo(mInst, axis, &togo);
```

```
int mcCtlGetInstanceHandle(MINSTANCE mInst, const char *owner,  
HMINSTANCE *hInst);
```

```
hInst, rc = mc.mcCtlGetInstanceHandle(  
    number mInst, string owner) or
```

```
hInst = mc.mcGetInstance(string owner)
```

```
MINSTANCE mInst=0;
```

```
HMINSTANCE hInst;
```

```
mcCtlGetInstanceHandle(mInst, "My Description", &hInst);  
mcCtlCycleStart(hInst); <font color="green">/Will show in the log as  
coming from "My Description"</font>
```

```
comment, rc = mc.mcCtlGetLocalComment(  
    number mInst,  
    number hVars)
```

```
<font color="green">-- Get local variables.</font> function m700(hVars)
```

```
    local inst = mc.mcGetInstance() <font color="green">-- Get the current  
    instance</font> local nilPoundVar = mc.mcCtlGetPoundVar(inst, 0) local  
    comment = <font color="blue">mc.mcCtlGetLocalComment</font>(inst,  
    hVars) local message = ""
```

```
    if hVars ~= nil then local flag, retval, rc flag, rc =  
        mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag == 1) then  
            retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_A) if rc ==  
                mc.MERROR_NOERROR then message = message .. "A" .. ":" ..  
                tostring(retval) .. ", "
```

```
    end
```

```
    end
```

```
    flag, rc = mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag ==  
        1) then retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_B) if rc ==  
            mc.MERROR_NOERROR then message = message .. "B" .. ":" ..  
            tostring(retval) end
```

```
end
```

```
mc.mcCtlSetLastError(inst, message) end
```

end

if (mc.mcInEditor() == 1) then

--We are testing the script in the editor.
--Fab up some paramters to pass to our m700()
function. local inst = mc.mcGetInstance() local hParams, rc = <font
color="blue">mc.mcCtlCreateLocalVars(inst, "A23 B6 (my
Comment)"--Like we are testing "M700 A23 B6 (my
Comment)". m700(hParams) -- Call m700
with hParams. end

```
retval, rc = mc.mcCtlGetLocalVar(  
    number mInst,  
    number hVars,  
    number varNumber)  
  
<font color="green">-- Get local variables.</font> function m700(hVars)  
  
    local inst = mc.mcGetInstance() -- Get the current instance local  
    nilPoundVar = mc.mcCtlGetPoundVar(inst, hVars, 0) local message = ""  
  
    if hVars ~= nil then local flag, retval, rc flag, rc =  
        mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag == 1) then  
            retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_A) if rc ==  
                mc.MERROR_NOERROR then message = message .. "A" .. ":" ..  
                tostring(retval) .. ", "  
  
        end  
  
    end  
  
    flag, rc = mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag ==  
        1) then retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_B) if rc ==  
            mc.MERROR_NOERROR then message = message .. "B" .. ":" ..  
            tostring(retval) end  
  
    end  
  
    mc.mcCtlSetLastError(inst, message) end
```

end

if (mc.mcInEditor() == 1) then

 m700(nil) -- We can't test this in the editor!
end

```

int mcCtlGetLocalVarFlag(
    MINSTANCE mInst, HMCVARS hVars,
    int varNumber,
    int *retval);

retval, rc = mc.mcCtlGetLocalVarFlag(
    number mInst,
    number hVars,
    number varNumber)

<font color="green">-- Get local variables.</font> function m700(hVars)
    local inst = mc.mcGetInstance() -- Get the current instance local
    nilPoundVar = mc.mcCtlGetPoundVar(inst,0) local message = ""

    if hVars ~= nil then local flag, retval, rc flag, rc =
        mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag == 1) then
        retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_A) if rc ==
        mc.MERROR_NOERROR then message = message .. "A" .. ":" ..
        tostring(retval) .. ", "
    end
    end

    flag, rc = mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag ==
    1) then retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_B) if rc ==
    mc.MERROR_NOERROR then message = message .. "B" .. ":" ..
    tostring(retval) end
end

```

```
mc.mcCtlSetLastError(inst, message) end
```

end

if (mc.mcInEditor() == 1) then

 m700(nil) -- We can't test this in the editor!
end

```
int mcCtlGetLogging(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mc.mcCtlGetLogging(  
    number mInst)  
  
<font color="green">// Check is logging is enabled.</font> MINSTANCE  
mInst = 0; BOOL enabled = FALSE; int rc = mcCtlGetLogging(mInst,  
&enabled); if (rc == MERROR_NOERROR && enabled == TRUE) {  
  
    <font color="green">// Logging is enabled!  
  
}  
  
</font>
```

mcCtlGetMachDir

C/C++ Syntax:

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

Usage:

```
//  
MINSTANCE mInst = 0;
```

```
int mcCtlGetMistDelay(  
    MINSTANCE mInst, double *secs);  
  
secs, rc = mcCtlGetMistDelay(  
    number mInst)  
  
<font color="green">// Get the mist delay.</font> MINSTANCE mInst = 0;  
  
double secs;  
  
int rc = mcCtlGetMistDelay(mInst, &secs); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetModalGroup(  
    MINSTANCE mInst, int group, double *val);  
  
val, rc = mc.mcCtlGetModalGroup(  
    number mInst, number group)  
  
<font color="green">// Get the modal code for modal group 1.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcCtlGetModalGroup(mInst, 1); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```



Home



Up a level



Previous



Next

mcCtlGetMode

```
int mcCtlGetOffset(  
    MINSTANCE mInst, int axisId, int type,  
    double *offset);  
  
offset, rc = mc.mcCtlGetOffset(  
    number mInst, number axisId, number type)  
  
<font color="green">// </font> MINSTANCE mInst = 0;
```

```
int mcCntlGetPoundVar(  
    MINSTANCE mInst, int param, double *value);  
  
value, rc = mc.mcCntlGetPoundVar(  
    number mInst, number param)  
  
int mInst=0;  
  
double PoundVar=50;  
  
double Value=0;  
  
<font color="green">;// Get the value of #50.</font>  
mcCntlGetPoundVar(mInst, PoundVar, &Value);
```

mcCtlGetRegister

C/C++ Syntax:

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

Usage:

```
//  
MINSTANCE mInst = 0;
```

```
int mcCtlGetRunTime(  
    MINSTANCE mInst, double *time);  
  
int mcCtlGetRunTime(MINSTANCE mInst, double *time)  
<font color="green">// Get the control run time in seconds.</font>  
MINSTANCE mInst = 0; double time;  
  
int rc = mcCtlGetRunTime(mInst, &time); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```



Home



Up a level



Previous



Next

mcCtlGetSpindleSpeed

C/C++ Syntax:

```
mcCtlSetSpindleSpeed(  
    MINSTANCE mInst,  
    double secs);
```

Notes: Not used at this time

END_FUNTION

```
int mcCtlGetState(  
    MINSTANCE mInst, mcState *state);  
  
mcState, rc = mc.mcCtlGetState(  
    MINSTANCE mInst)  
  
<font color="green">// Get the state of controller instance 0.</font>  
MINSTANCE mInst = 0; mcState state;  
  
char stateName[80]; int rc = <font color="blue">mcCtlGetState(mInst,  
&state)</font>; if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> rc =  
    mcCtlGetStateName(mInst, state, stateName, sizeof(stateName)); }
```

```
int mcCtlGetName(
    MINSTANCE mInst, mcState state, char *buf, size_t bufSize);

buf, rc = mcCtlGetName(
    number mInst, number state)

<font color="green">// Get the state name for controller instance 0.</font>
MINSTANCE mInst = 0;

mcState state;

char stateName[80];

int rc = mcCtlGetName(mInst, &state); if (rc == MERROR_NOERROR) {

    <font color="green">// Success!</font> rc = <font
color="blue">mcCtlGetName(mInst, state, stateName,
sizeof(stateName))</font>; }
```

```
int mcCtlGetStats(
```

```
    MINSTANCE mInst, mstats_t *stats);
```

N/A

```
struct mstats {
```

```
    int cannon_buf_depth; int la_cannon_buf_depth; double totalTime;  
    double sessionTime; double spindleTime; long m3count; long m4count;  
    long m6count; double xDistance; double yDistance; double zDistance;  
    double aDistance; double bDistance; double cDistance; };
```

```
typedef struct mstats mstats_t;
```

```
<font color="green">// Get controller statistics.</font> MINSTANCE mInst  
= 0; mstats_t stats;
```

```
int rc = mcCtlGetStats(mInst, &stats); if (rc == MERROR_NOERROR) {
```

```
    printf("M3 count = %dn", stats.m3Count); }
```

```
int mcCtlGetToolOffset(  
    MINSTANCE mInst, int axisId, double *offset);  
  
offset, rc = mc.mcCtlGetToolOffset(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int toolnum = 5; double val = 0; mcCtlGetToolOffset(mInst, Z_AXIS,  
&val); <font color="green">// Get the tool offset distance for the Z axis..  
</font>
```

```
int mcCtlGetUnitsCurrent(  
    MINSTANCE mInst, int *units);  
  
units, rc = mc.mcCtlGetUnitsCurrent(  
    number mInst)  
  
<font color="green">// Get the current machine units.</font>  
MINSTANCE mInst = 0;  
  
int units;  
  
int rc = mcCtlGetUnitsCurrent(mInst, &units); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetUnitsDefault(  
    MINSTANCE mInst, int *units);  
  
units, rc = mc.mcCtlGetUnitsDefault(  
    number mInst)  
  
<font color="green">// Get the default machine units.</font> MINSTANCE  
mInst = 0;  
  
int units;  
  
int rc = mcCtlGetUnitsDefault(mInst, &units); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> }
```

```
int mcCtlGetValue(  
    MINSTANCE mInst, int valId, int param, double *value);  
  
value, rc = mc.mcCtlGetValue(  
    number mInst, number valId, number param)  
  
int mInst = 0;  
  
int motor = 1;  
  
double value = 0;  
  
<font color="green">// Get the motor velocity of motor1.</font> int rc =  
mcCtlGetValue(mInst, VAL_MOTOR_VEL, motor, &value);
```

```
int mcCtlGetVersion(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCtlGetVersion(  
    number mInst)  
  
<font color="green">// Get the core version string.</font> MINSTANCE  
mInst = 0; char ver[80];  
  
int rc = mcCtlGetVersion(mInst, ver, sizeof(ver)); if (rc ==  
MERROR_NOERROR) {  
  
    printf("The core version is %sn", ver); }
```

```
int mcCtlIsStill(  
    MINSTANCE mInst, BOOL *still);  
  
still, rc = mcCtlIsStill(  
    number mInst)  
  
<font color="green">// See if the control axes are still.</font>  
MINSTANCE mInst = 0;  
  
BOOL still;  
  
int rc = mcCtlIsStill(mInst, &still); if (rc == MERROR_NOERROR) {  
    if (still == TRUE) {  
  
        <font color="green">// All axes are still.</font> } else {  
  
        <font color="green">// At least one axis is moving!</font> }  
    }  
}
```

```
int mcCtlMachineStateClear(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlMachineStateClear(  
    number mInst)  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); <font color="green">-- push control  
    state to the stack.</font> mc.mcCtlMachineStatePush(inst); <font  
    color="green">-- put machine in G20, and G90 mode.</font>  
    mc.mcCtlGcodeExecuteWait(inst, "G20nG90"); <font color="green">--  
    push control state to the stack.</font> mc.mcCtlMachineStatePush(inst);  
    <font color="green">-- put machine in G21, and G91 mode.</font>  
    mc.mcCtlGcodeExecuteWait(inst, "G21nG91"); <font color="green">--  
    clear the machine state stack and leave the machine in G21 and G91 modes.  
    </font> mc.mcCtlMachineStateClear(inst); end
```

```
int mcCtlMachineStatePop(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlMachineStatePop(  
    number mInst)  
  
<font color="green">-- LUA example</font> function test()  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); <font color="green">-- push control  
    state to the stack saving original modes.</font>  
    mc.mcCtlMachineStatePush(inst); <font color="green">-- put machine in  
    G20, and G90 mode.</font> mc.mcCtlGcodeExecuteWait(inst,  
    "G20nG90"); <font color="green">-- push control state to the stack.</font>  
    mc.mcCtlMachineStatePush(inst); <font color="green">-- put machine in  
    G21, and G91 mode.</font> mc.mcCtlGcodeExecuteWait(inst,  
    "G21nG91"); <font color="green">-- restore the machine state stack to G20  
    and G90 modes.</font> mc.mcCtlMachineStatePop(inst); <font  
    color="green">-- restore the machine state stack to original modes.</font>  
    mc.mcCtlMachineStatePop(inst); end
```

```
int mcCtlMachineStatePush(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlMachineStatePush(  
    number mInst)  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); <font color="green">-- push control  
    state to the stack saving original modes.</font>  
    mc.mcCtlMachineStatePush(inst); <font color="green">-- put machine in  
    G20, and G90 mode.</font> mc.mcCtlGcodeExecuteWait(inst,  
    "G20nG90"); <font color="green">-- push control state to the stack.</font>  
    mc.mcCtlMachineStatePush(inst); <font color="green">-- put machine in  
    G21, and G91 mode.</font> mc.mcCtlGcodeExecuteWait(inst,  
    "G21nG91"); <font color="green">-- restore the machine state stack to G20  
    and G90 modes.</font> mc.mcCtlMachineStatePop(inst); <font  
    color="green">-- restore the machine state stack to original modes.</font>  
    mc.mcCtlMachineStatePop(inst); end
```

```
int mcCtlMacroAlarm(  
    MINSTANCE mInst int error, const char *message);  
  
rc = mc.mcCtlMacroAlarm(  
    number mInst number error string message)  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); mc.mcCtlMacroAlarm(inst, 19, "Test  
    Alram") end
```

```
int mcCtlMacroStop(  
    MINSTANCE mInst int error, const char *message);  
  
rc = mc.mcCtlMacroStop(  
    number mInst number error string message)  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); mc.mcCtlMacroStop(inst, 19, "Test  
    Stop") end
```

```
int mcCtlMdiExecute(  
    MINSTANCE mInst, const char *commands);  
  
rc = mc.mcCtlMdiExecute(  
    number mInst, string commands)  
  
MINSTANCE mInst = 0;  
  
int rc;  
  
<font color="green">// Move the machine back to xy then z zero.</font>  
<font color="green">// Correct example.</font> rc =  
mcCtlMdiExecute(mInst, "G00 G90 X0.0 Y0.0\nZ0.0");  
  
<font color="green">// Incorrect example.</font> rc =  
mcCtlMdiExecute(mInst, "G00 G90 X0.0 Y0.0"); rc =  
mcCtlMdiExecute(mInst, "Z0.0");
```

mcCntlPluginConfig

C/C++ Syntax:

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

Usage:

```
//  
MINSTANCE mInst = 0;
```

mcCntlPluginDiag

C/C++ Syntax:

LUA Syntax:

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

Usage:

```
//  
MINSTANCE mInst = 0;
```

```
int mcCtlProbeFileClose(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlProbeFileClose(  
    number mInst)  
  
<font color="green">-- probe file close LUA macro example.</font>  
function m112()  
    local inst=mc.mcGetInstance(); local rc =  
    mc.mcCtlProbeFileClose(inst); end  
  
  
  
if (mc.mcInEditor() == 1) then m112() end
```

```
int mcCtlProbeFileOpen(  
    MINSTANCE mInst, const char *fileName, const char *format, BOOL  
    overWrite);  
  
rc = mc.mcCtlProbeFileOpen(  
    number mInst, string fileName, string format, number overWrite);  
  
<font color="green">-- probe file open LUA macro example.</font>  
function m111()  
  
    local inst = mc.mcGetInstance(); local rc =  
    mc.mcCtlProbeFileOpen(inst, 'probetest.csv', '%.4AXIS_X, %.4AXIS_Y,  
    %.4AXIS_Z\r\n', TRUE); end
```

```
if (mc.mcInEditor() == 1) then m111() end
```

```
int mcCtlProbeGetStrikeStatus(  
    MINSTANCE mInst, BOOL *didStrike);  
  
didstrike, rc = mc.mcCtlProbeGetStrikeStatus(  
    number mInst)  
  
<font color="green">-- probe get strike status example.</font>  
  
local inst = mc.mcGetInstance()  
  
local didStrike, rc = mc.mcCtlProbeGetStrikeStatus(inst)
```

```
int mcCtlSetCoolantDelay(  
    MINSTANCE mInst, double secs);  
  
rc = mc.mcCtlSetCoolantDelay(  
    number mInst, number secs);  
  
<font color="green">// Set 3/4 second coolant delay.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcCtlSetCoolantDelay(mInst, .750);
```

```
int mcCtlSetDiaMode(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mcCtlSetDiaMode(  
    number mInst, number enable);  
  
<font color="green">// Enable diamter mode</font> MINSTANCE mInst =  
0; int rc = mcCtlSetDiaMode(mInst, TRUE);
```

```
int mcCtlSetMistDelay(  
    MINSTANCE mInst, double secs);  
  
rc = mcCtlSetMistDelay(  
    number mInst, number secs);  
  
<font color="green">// Set the mist delay to 3/4 of a second.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcCtlSetMistDelay(mInst, .750);
```

```
int mcCtlSetMode(  
    MINSTANCE mInst, double mode);  
  
rc = mc.mcCtlSetMode(  
    number mInst, number mode)  
  
<font color="green">// Set the interpreter mode to Lathe Diameter.</font>  
MINSTANCE mInst = 0; int rc = mcCtlSetMode(mInst,  
    MC_MODE_LATHE_DIA);
```

mcCtlSetPoundVar(

MINSTANCE mInst, int param, double value);

mcCtlSetPoundVar(

number mInst, number param, number value)

MINSTANCE mInst = 0; int PoundVar = 50;

double Value=21;

// Set the value of #50 to 21. int rc =
mcCtlSetPoundVar(mInst, PoundVar, Value);

```
int mcCtlSetStats(  
    MINSTANCE mInst, mstats_t *stats);
```

N/A

```
<font color="green">// </font> MINSTANCE mInst = 0;
```

```
int mcCtlSetValue(  
    MINSTANCE mInst, int valId, int param, double value);  
  
rc = mcCtlSetValue(  
    number mInst, number valId, number param, number value)  
    MINSTANCE mInst = 0; int motor = 1;  
  
value=1000.0;  
  
<font color="green">// Set the Velocity of the motor to 1000.0 .</font> in  
rc = mcCtlSetValue(mInst, VAL_MOTOR_VEL, motor, value);
```

```
int mcFileHoldAquire(  
    MINSTANCE mInst, const char *reason, int JogAxisBits);  
  
rc = mc.mcFileHoldAquire(  
    number mInst, string reason, number JogAxisBits)  
  
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance();  
  
local rc = mc.mcFileHoldAquire(inst, "My hold reason", 0);
```

```
int mcFileHoldReason(
```

```
    MINSTANCE mInst, char *buf, long bufSize);
```

```
reason, rc = mc.mcFileHoldReason(
```

```
    number mInst)
```

```
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance(); local rc = mc.mcFileHoldAquire(inst, "My hold  
reason", 0); loacl reason
```

```
reason, rc = mc.mcFileHoldReason(inst);
```

```
int mcFileHoldRelease(  
    MINSTANCE mInst);  
  
rc = mc.mcFileHoldRelease(  
    number mInst)  
  
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance(); local rc = mc.mcFileHoldAquire(inst, "My hold  
reason", 0); loacl reason  
  
reason, rc = mc.mcFileHoldReason(inst); <font color="green">-- Do some  
script magic...</font> rc = mc.mcFileHoldRelease(inst); <font  
color="green">-- G code processing resumes.</font>
```



Home



Up a level



Previous



Next

Gcode File

- [mcCtlGetGcodeLine](#)
- [mcCtlGetGcodeLineCount](#)
- [mcCtlGetGcodeLineNbr](#)
- [mcCtlRewindFile](#)
- [mcCtlSetGcodeLineNbr](#)
- [mcCtlLoadGcodeFile](#)
- [mcCtlLoadGcodeString](#)
- [mcCtlCloseGcodeFile](#)
- [mcCtlGetGcodeFileName](#)

```
int mcCtlGetGcodeLine(  
    MINSTANCE mInst, int LineNumber, char *buf, long bufSize);  
  
buff, rc = mc.mcCtlGetGcodeLine(  
    number mInst, number LineNumber)  
  
int mInst=0;  
  
int LineNumber = 5; char gline[128];  
  
gline[0] = '0';  
  
mcCtlGetGcodeLine(mInst, LineNumber, gline , 128); <font  
color="green">// Get line number 5 from the Gcode file loaded into  
controller 0.</font>
```

```
int mcCtlGetGcodeLineCount(  
    MINSTANCE mInst, double *count);  
  
count, rc = mc.mcCtlGetGcodeLineCount(  
    number mInst)  
  
int mInst=0;  
  
double count=0; mcCtlGetGcodeLineCount(mInst, &count);
```

```
int mcCtlGetGcodeLineNbr(  
    MINSTANCE mInst, double *val);  
  
val, rc = mc.mcCtlGetGcodeLineNbr(  
    number mInst)  
  
int mInst=0;  
  
double dline=0; mcCtlGetGcodeLineNbr(mInst, &dline); <font  
color="green">//Get the current running line number</font>
```

```
int mcCtlRewindFile(
```

```
    MINSTANCE mInst);
```

```
rc = mc.mcCtlRewindFile(
```

```
    number mInst)
```

```
    MINSTANCE mInst = 0; mcCtlRewindFile(mInst); <font  
color="green">// Rewind controller 0's Gcode file.</font>
```

```
int mcCtlSetGcodeLineNbr(
```

```
    MINSTANCE mInst, double line);
```

```
int mInst=0; mcCtlSetGcodeLineNbr(mInst, 10); <font color="green">//  
Set the Gcode file to be at line 10.</font>
```

```
int mcCtlLoadGcodeFile(  
    MINSTANCE mInst, const char *FileToLoad);  
  
rc = mc.mcCtlLoadGcodeFile(  
    number mInst, stringFileToLoad)  
  
int mInst=0;  
  
char File[128] = "C:SomeGocdefile.tap"; int rc =  
mcCtlLoadGcodeFile(mInst, &char); <font color="green">// Load  
SomeGocdefile.tap file.</font> if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> }
```

```
int mcCtlLoadGcodeString(  
    MINSTANCE mInst, const char *gCode);  
  
rc = mc.mcCtlLoadGcodeString(  
    MINSTANCE mInst, const char *gCode);  
  
<font color="green">// Load G code from a string.</font> MINSTANCE  
mInst = 0;  
  
char *gCode = "%nO1001nG90 G94 G91.1 G40 G49 G17nG20"  
  
int rc = mcCtlLoadGcodeString(mInst, gCode);
```

```
int mcCtlCloseGCodeFile(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlCloseGCodeFile(  
    number mInst)  
  
int mInst=0;  
  
mcCtlCloseGCodeFile(mInst); <font color="green">// Close the Gcode  
file in controller 0.</font>
```

```
int mcCtlGetGcodeFileName(
    MINSTANCE mInst, char *buf, size_t bufSize);

buf, rc = mc.mcCtlGetGcodeFileName(
    number mInst)

<font color="green">// Get the currently loaded G code file name.</font>
MINSTANCE mInst = 0;

char gCodeFileName[255];

int rc = mcCtlGetGcodeFileName(mInst, gCodeFileName,
    sizeof(gCodeFileName)); if (rc == MERROR_NOERROR) {

    <font color="green">// Success.</font> }
```

Tools

- [mcToolGetCurrent](#)
- [mcToolGetData](#)
- [mcToolGetDataExDbl](#)
- [mcToolGetDataExInt](#)
- [mcToolGetDataExStr](#)
- [mcToolGetDesc](#)
- [mcToolGetSelected](#)
- [mcToolLoadFile](#)
- [mcToolSaveFile](#)
- [mcToolSetCurrent](#)
- [mcToolSetData](#)
- [mcToolSetDataExDbl](#)
- [mcToolSetDataExInt](#)
- [mcToolSetDataExStr](#)
- [mcToolSetDesc](#)

```
int mcToolGetCurrent(  
    MINSTANCE mInst, int *toolnum);  
  
toolnum, rc = mc.mcToolGetCurrent(  
    number mInst)  
  
<font color="green">// Get the current tool.</font> MINSTANCE mInst =  
0; int toolnum = 0;  
  
int rc = mcToolGetCurrent(mInst, &toolnum);
```

```
int mcToolGetData(  
    MINSTANCE mInst, int type, int toolNumber, double *value);  
  
value, rc = mc.mcToolGetData(  
    number mInst, number type, number toolNumber)  
  
MINSTANCE mInst = 0; int toolnum = 5;  
  
double val = 0;  
  
<font color="green">// Get the tool height offset for tool number 5.</font>  
int rc = mcToolGetData(mInst, MTOOL_MILL_HEIGHT, toolnum, &val);
```

```
int mcToolGetDataExDbl(
    MINSTANCE mInst, int toolnum, const char *fieldName, double
    *value);

value, rc = mc.mcToolGetDataExDbl(
    number mInst, number toolNumber, string fieldName)

MINSTANCE mInst = 0;

int toolnum = 5;

double val = 0;

<font color="green">// Get the contents of "MyUserField" for tool number
5.</font> int rc = mcToolGetDataExDbl(mInst, toolnum, "MyUserField",
&val);
```

```
int mcToolGetDataExInt(  
    MINSTANCE mInst, int toolnum, const char *fieldName, int *value);  
  
value, rc = mc.mcToolGetDataExInt(  
    number mInst, number toolNumber, string fieldName)  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
int val = 0;  
  
<font color="green">// Get the contents of "MyUserField" for tool number  
5.</font> int rc = mcToolGetDataExInt(mInst, toolnum, "MyUserField",  
&val);
```

```
int mcToolGetDataExStr(  
    MINSTANCE mInst, int toolnum, const char *fieldName, char *value,  
    size_t valueLen);  
  
value, rc = mc.mcToolGetDataExStr(  
    number mInst, number toolNumber, string fieldName)  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
char val[128];  
  
<font color="green">// Get the contents of "MyUserField" for tool number  
5.</font> int rc = mcToolGetDataExStr(mInst, toolnum, "MyUserField",  
val, sizeof(val));
```

```
int mcToolGetDesc(  
    MINSTANCE mInst, int toolnum, char *buff, size_t bufsize);  
  
buff, rc = mc.mcToolGetDesc(  
    number mInst, number toolnum)  
  
MINSTANCE mInst = 0; char desc[128];  
  
int toolnum = 5;  
  
<font color="green">// Get the description of tool number 5 for controller  
0.</font> int rc = mcToolGetDesc(mInst, toolnum, desc, sizeof(desc));
```

```
int mcToolGetSelected(  
    MINSTANCE mInst, int *toolnum);  
  
toolnum, rc = mc.mcToolGetSelected(  
    number mInst)  
  
<font color="green">// Get the selected tool number.</font> MINSTANCE  
mInst = 0;  
  
int toolnum = 0;  
  
int rc = mcToolGetSelected(mInst, &toolnum);
```

```
int mcToolLoadFile(  
    MINSTANCE mInst, const char *FileToLoad);  
  
rc = mc.mcToolLoadFile(  
    number mInst, string FileToLoad)  
  
<font color="green">// Load a tool table.</font> MINSTANCE mInst = 0;  
  
int rc = mcToolLoadFile(mInst, "tooltable.xls");
```

```
int mcToolSaveFile(  
    MINSTANCE mInst);
```

```
rc = mcToolSaveFile(  
    number mInst)
```

```
<font color="green">// Save the previously loaded tool table file.</font>  
MINSTANCE mInst = 0; int rc = mcToolSaveFile(mInst);
```

```
int mcToolSetCurrent(  
    MINSTANCE mInst, int toolnum);  
  
rc = mc.mcToolSetCurrent(  
    number mInst, number toolnum)  
  
<font color="green">// Set the current tool to tool #1.</font> MINSTANCE  
mInst = 0; int rc = mcToolSetCurrent(mInst, 1);
```

```
int mcToolSetData(  
    MINSTANCE mInst, int Type, int Toolnumber, double val);  
  
rc = mc.mcToolSetData(  
    number mInst, number Type, number Toolnumber, number val);  
  
MINSTANCE mInst = 0; int toolnum = 5;  
  
double val = 0;  
  
<font color="green">// Set the tool height wear offset to zero.</font> int rc  
= mcToolSetData(mInst, MTOOL_MILL_HEIGHT_W, toolnum, val);
```

```
int mcToolSetDataExDbl(  
    MINSTANCE mInst, int toolNum, const char *fieldName, double value);  
  
rc = mc.mcToolSetDataExDbl(  
    number mInst, number toolNum, string fieldName, number val);  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
double val = 0;  
  
<font color="green">// Set the user field "MyUserField" to zero.</font> int  
rc = mcToolSetDataExDbl(mInst, toolnum, "MyUserField", val);
```

```
int mcToolSetDataExInt(  
    MINSTANCE mInst, int toolNum, const char *fieldName, int value);  
  
rc = mc.mcToolSetDataExInt(  
    number mInst, number toolNum, string fieldName, number val);  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
int val = 0;  
  
<font color="green">// Set the user field "MyUserField" to zero.</font> int  
rc = mcToolSetDataExInt(mInst, toolnum, "MyUserField", val);
```

```
int mcToolSetDataExStr(  
    MINSTANCE mInst, int toolNum, const char *fieldName, const char  
    *value);  
  
rc = mc.mcToolSetDataExStr(  
    number mInst, number toolNum, string fieldName, string val);  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
char val[128];  
  
<font color="green">// Set the user field "MyUserField" to zero.</font>  
strcpy(val, "0");  
  
int rc = mcToolSetDataExStr(mInst, toolnum, "MyUserField", val);
```

```
int mcToolSetDesc(  
    MINSTANCE mInst, int toolnum, const char *tdsc);  
  
rc = mc.mcToolSetDesc(  
    number mInst, number toolnum, string tdsc);  
  
MINSTANCE mInst = 0;  
  
char desc[128] = "My Best 1/2 inch endmill"; int toolnum = 5;  
  
<font color="green">// Set the description of tool number 5.</font> int rc =  
mcToolSetDesc(mInst, toolnum, desc);
```

Jogging

- [mcJogAbsStart](#)
- [mcJogAbsStop](#)
- [mcJogGetAccel](#)
- [mcJogGetFeedRate](#)
- [mcJogGetFollowMode](#)
- [mcJogGetInc](#)
- [mcJogGetRate](#)
- [mcJogGetUnitsMode](#)
- [mcJogGetVelocity](#)
- [mcJogIncStart](#)
- [mcJogIncStop](#)
- [mcJogIsJogging](#)
- [mcJogIsStopping](#)
- [mcJogSetAccel](#)
- [mcJogSetFeedRate](#)
- [mcJogSetFollowMode](#)
- [mcJogSetInc](#)
- [mcJogSetRate](#)
- [mcJogSetTraceEnable](#)
- [mcJogSetType](#)
- [mcJogSetUnitsMode](#)
- [mcJogVelocityStart](#)
- [mcJogVelocityStop](#)

```
int mcJogAbsStart(  
    MINSTANCE mInst, int axisId, double pos);  
  
rc = mc.mcJogAbsStart(  
    number mInst, number axisId, number pos)  
  
MINSTANCE mInst = 0;  
  
int axis = Z_AXIS;  
  
double JogToPos = 5.0; <font color="green">// Jog controller 0 to position  
5.0 in the Z axis.</font> mcJogAbsStart(mInst, axis, Jogpos);
```

```
int mcJogAbsStop(  
    MINSTANCE mInst, int axisId, double incr);  
  
rc = mc.mcJogAbsStop(  
    number mInst, number axisId, number incr)  
  
<font color="green">// Stop the incremental jog on the Z axis.</font>  
MINSTANCE mInst = 0; int axis = Z_AXIS;  
  
double Joginc = .100; int rc = mcJogAbsStop(mInst, axis, Joginc);
```

```
int mcJogGetAccel(  
    MINSTANCE mInst, int axisId, double *percent);  
  
percent, rc = mc.mcJogGetAccel(  
    number mInst, number axis)  
  
<font color="green">// Get the jog accel percentage for the X axis.</font>  
MINSTANCE mInst = 0;  
  
double accel;  
  
int rc = mcJogGetAccel(mInst, X_AXIS, &accel);
```

```
int mcJogGetFeeRate(  
    MINSTANCE mInst, int axisId, double *feedRate);  
  
feedRate, rc = mc.mcJogSetFeedRate(  
    number mInst, number axisId)  
  
<font color="green">// Set the jog rate to 75% of the Z axis maximum  
velocity.</font> MINSTANCE mInst = 0;  
  
int axis = Z_AXIS;  
  
double feedRate = 0.0;  
  
int rc = mcJogSetUnitsMode(mInst, axis, MC_UNITS_INCH); rc =  
mcJogSetFeedRate(mInst, axis, &feedRate); <font color="green">// Get the  
feed rate in inches per minute.</font>
```

```
mcJogGetFollowMode(  
    MINSTANCE mInst, double *mode_on);  
  
int mInst=0;  
  
double jfstat=0; mcJogGetFollowMode(mInst, &jfstate);
```

```
int mcJogGetInc(  
    MINSTANCE mInst, int axisId, double *increment);  
  
increment, rc = mc.mcJogGetInc(  
    number mInst, number axisId)  
  
<font color="green">// </font> MINSTANCE mInst = 0;
```

```
int mcJogGetRate(  
    MINSTANCE mInst, int axisId, double *percent);  
  
percent, rc = mc.mcJogGetRate(  
    number mInst, number axisId)  
  
<font color="green">// Get the current jog rate for the X axis</font>  
MINSTANCE mInst = 0;  
  
double jogRate;  
  
int rc = mcJogGetRate(mInst, X_AXIS, &jogRate);
```

```
int mcJogGetTraceEnable(MINSTANCE mInst, BOOL *enable);  
enable, rc = mc.mcJogGetTraceEnable(number mInst)  
<font color="green">// Get jog trace enable status.</font> MINSTANCE  
mInst = 0;  
  
int enabled = MC_FALSE; int rc = mcJogGetTraceEnable(mInst,  
&enabled);
```

```
int mcJogGetType(MINSTANCE mInst, int axisId, int *type);

type, rc = mc.mcJogGetType(
    number mInst, number axisId)

<font color="green">// Get the X axis jog type. </font> MINSTANCE
mInst = 0;

int type = 0;

int rc = mcJogGetType(mInst, X_AXIS, &type);
```

```
mcJogGetUnitsMode(
```

```
    MINSTANCE mInst, int axisId, int *mode);
```

```
int mInst = 0; int mode = 0; mcJogGetUnitsMode(mInst, X_AXIS,  
    &mode);
```

```
int mcJogGetVelocity(  
    MINSTANCE mInst, int axisId, double *vel);  
  
vel, rc = mc.mcJogGetVelocity(  
    number mInst, number axisId),  
  
<font color="green">// Get the current jog velocity setting for the X  
axis</font> MINSTANCE mInst = 0; double jogVel;  
  
int rc = mcJogGetVelocity(mInst, X_AXIS, &jogVel);
```

```
int mcJogIncStart(  
    MINSTANCE mInst, int axisId, double dist);  
  
rc = mc.mcJogIncStart(  
    number mInst, number axisId, number dist)  
  
MINSTANCE mInst = 0; int axis = Z_AXIS;  
  
double Joginc = .100; <font color="green">// Jog controller 0 .1 in the Z  
axis.</font> mcJogIncStart(mInst, axis, Joginc);
```

```
int mcJogIncStop(  
    MINSTANCE mInst, int axisId, double incr);  
  
rc = mc.mcJogIncStop(  
    number mInst, number axisId, number incr)  
  
<font color="green">// Stop the incremental jog on the Z axis.</font>  
MINSTANCE mInst = 0; int axis = Z_AXIS;  
  
double Joginc = .100; int rc = mcJogIncStop(mInst, axis, Joginc);
```

```
int mcJogIsJogging(  
    MINSTANCE mInst, int axisId, BOOL *jogging);  
  
jogging, rc = mc.mcJogIsJogging(  
    number mInst, number axisId)  
  
<font color="green">// See if the X axis is jogging.</font> MINSTANCE  
mInst = 0;  
  
BOOL jogging = FALSE;  
  
int rc = mcJogIsJogging(mInst, X_AXIS, &jogging);
```

```
int mcJogIsStopping(
    MINSTANCE mInst, int axisId, BOOL *stopping);
    stopping, rc = mc.mcJogIsStopping(
        number mInst, number axisId)

<font color="green">// See if the X axis is stopping.</font> MINSTANCE
mInst = 0;

BOOL stopping = FALSE;
int rc = mcJogIsStopping(mInst, X_AXIS, &stopping);
```

```
int mcJogSetAccel(  
    MINSTANCE mInst, int axisId, double percent);  
  
rc = mc.mcJogSetAccel(  
    number mInst, number axisId, double percent)  
  
<font color="green">// Set the X axis jog accel percentage to 75%.</font>  
MINSTANCE mInst = 0;  
  
int mcJogSetAccel(mInst, X_AXIS, 75);
```

```
int mcJogSetFeeRate(  
    MINSTANCE mInst, int axisId, double feedRate);  
  
rc = mc.mcJogSetFeedRate(  
    number mInst, number axisId, number feedRate)  
  
<font color="green">// Set the jog rate to 75% of the Z axis maximum  
velocity.</font> MINSTANCE mInst = 0;  
  
int axis = Z_AXIS;  
  
int rc = mcJogSetUnitsMode(mInst, axis, MC_UNITS_INCH); rc =  
mcJogSetFeedRate(mInst, axis, 20); <font color="green">// 20 inches per  
minute.</font>
```

```
mcJogSetFollowMode(  
    MINSTANCE mInst, double mode_on);  
  
int mInst=0; mcJogGetFollowMode(mInst, MC_ON);
```

```
int mcJogSetInc(  
    MINSTANCE mInst, int axisId, double increment);  
  
rc = mc.mcJogSetInc(  
    number mInst, number axisId, number increment)  
  
<font color="green">// Set the X axis jog increment to .010</font>  
MINSTANCE mInst = 0;  
  
int rc = mcJogSetInc(mInst, X_AXIS, .010);
```

```
int mcJogSetRate(  
    MINSTANCE mInst, int axisId, double percent);  
  
rc = mc.mcJogSetRate(  
    number mInst, number axisId, number percent)  
  
<font color="green">// Set the jog rate to 75% of the Z axis maximum  
velocity.</font> MINSTANCE mInst = 0; int axis = Z_AXIS;  
  
int rc = mcJogSetRate(mInst, axis, 75);
```

```
int mcJogSetTraceEnable(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcJogSetTraceEnable(  
    number mInst, number enable)  
  
<font color="green">//Set jog tracing.</font> MINSTANCE mInst = 0;  
  
int rc = mcJogSetTraceEnable(mInst, MC_TRUE);
```

```
int mcJogSetType(MINSTANCE mInst, int axisId, int type);  
rc = mc.mcJogSetType(  
    number mInst, number axisId, number type)  
<font color="green">// Set the X axis jog type to velocity mode. </font>  
MINSTANCE mInst = 0;  
int rc = mcJogSetType(mInst, X_AXIS, MC_JOG_TYPE_VEL);
```

```
mcJogSetUnitsMode(  
    MINSTANCE mInst, int axisId, int mode);  
  
int mInst = 0;  
  
mcJogSetUnitsMode(mInst, X_AXIS, MC_UNITS_INCH);
```

```
int mcJogVelocityStart(  
    MINSTANCE mInst, int axisId, double dir);  
  
rc = mc.mcJogVelocityStart(  
    number mInst, number axisId, number dir);  
  
int mInst=0;  
  
int axis = Z_AXIS;  
  
int rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS); <font  
color="green">// Start Z axis jogging for 5 sec.</font> if (rc ==  
MERROR_NOERROR) {  
  
    Sleep(5000); }  
  
rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS); <font  
color="green">// Reverse axis and jog the axis Back.</font> if (rc ==  
MERROR_NOERROR) {  
  
    Sleep(5000); }  
  
mcJogVelocityStop(mInst, axis); <font color="green">// Stop the axis.  
</font>
```

```
mcJogVelocityStop(
```

```
    MINSTANCE mInst, int axisId);
```

```
mcJogVelocityStop(
```

```
    number mInst, number axisId)
```

```
MINSTANCE mInst = 0; int axis = Z_AXIS;
```

```
int rc;
```

```
<font color="green">// Jog axis at 10 % max velocity.</font> rc =  
mcJogSetRate(mInst, axis, 10); rc = mcJogVelocityStart(mInst, axis,  
MC_JOG_POS); if (rc == MERROR_NOERROR) {
```

```
    Sleep(5000); }
```

```
<font color="green">// Stop the axis.</font> rc =  
mcJogVelocityStop(mInst, axis);
```

MPGs

- [mcMpgGetAccel](#)
- [mcMpgGetAxis](#)
- [mcMpgGetCountsPerDetent](#)
- [mcMpgGetEnable](#)
- [mcMpgGetEncoderReg](#)
- [mcMpgGetInc](#)
- [mcMpgGetReversed](#)
- [mcMpgGetRate](#)
- [mcMpgGetShuttleMode](#)
- [mcMpgGetShuttlePercent](#)
- [mcMpgMoveCounts](#)
- [mcMpgSetAccel](#)
- [mcMpgSetAxis](#)
- [mcMpgSetCountsPerDetent](#)
- [mcMpgSetEnable](#)
- [mcMpgSetEncoderReg](#)
- [mcMpgSetInc](#)
- [mcMpgSetReversed](#)
- [mcMpgSetRate](#)
- [mcMpgSetShuttleMode](#)
- [mcMpgSetShuttlePercent](#)

```
int mcMpgGetAccel(  
    MINSTANCE mInst, int mpg, double *percentMaxAccel);  
  
percentMaxAccel, rc = mc.mcMpgGetAccel(  
    number mInst, number mpg)  
  
<font color="green">// Get the current accel for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
double percentMaxAccel = 0; int rc;  
  
rc = mcMpgGetAccel(mInst, 0, &percentMaxAccel);
```

```
int mcMpgGetAxis(  
    MINSTANCE mInst, int mpg, int *axisId);  
  
axisId, rc = mc.mcMpgGetAxis(  
    number mInst, number mpg)  
  
<font color="green">// Get the currently selected axis for MPG 0.</font>  
MINSTANCE mInst = 0; int axisId;  
  
int rc = mcMpgGetAxis(mInst, 0, &axisId);
```

```
int mcMpgGetCountsPerDetent(  
    MINSTANCE mInst, int mpg, int *pulses);  
  
pulses, rc = mc.mcMpgGetCountsPerDetent(  
    number mInst, number mpg)  
  
<font color="green">// Get the counts per detent for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
int cnts;  
  
int rc = mcMpgGetCountsPerDetent(mInst, 0, &cnts);
```

```
int mcMpgGetEnable(  
    MINSTANCE mInst, int mpg, BOOL *enabled);  
  
enabled, rc = mc.mcMpgGetEnable(  
    number mInst, number mpg)  
  
<font color="green">// Get the enabled status of MPG 0.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = MC_FALSE;  
  
  
  
int rc = mcMpgGetEnable(mInst, 0, &enabled);
```

```
int mcMpgGetEncoderReg(  
    MINSTANCE mInst, int mpg, HMCREG *hReg);  
  
hReg, rc = mc.mcMpgGetEncoderReg(  
    number mInst, number mpg)  
  
<font color="green">// Get the current encoder associated with MPG 0.  
</font> MINSTANCE mInst = 0;  
  
HMCREG hReg = 0;
```

```
int rc = mcMpgGetEncoderReg(mInst, 0, &hReg);
```

```
int mcMpgGetInc(  
    MINSTANCE mInst, int mpg, double *inc);  
  
inc, rc = mc.mcMpgGetInc(  
    number mInst, number mpg)  
  
<font color="green">// Get the incremnt for MPG 0.</font> MINSTANCE  
mInst = 0; double inc;  
  
int rc = mcMpgGetInc(mInst, 0, &inc);
```

```
int mcMpgGetReversed(  
    MINSTANCE mInst, int mpg, BOOL *reversed);  
  
reversed, rc = mc.mcMpgGetReversed(  
    number mInst, number mpg)  
  
<font color="green">// Get the reversed status of MPG 0.</font>  
MINSTANCE mInst = 0;  
  
BOOL reversed = MC_FALSE;  
  
int rc = mcMpgGetReversed(mInst, 0, &reversed);
```

```
int mcMpgGetRate(  
    MINSTANCE mInst, int mpg, double *percentMaxVel);  
  
percentMaxVel, rc = mc.mcMpgGetRate(  
    number mInst, number mpg)  
  
<font color="green">// Get the velocity percentage for MPG 0.</font>  
MINSTANCE mInst = 0; double velPercent;  
  
int rc = mcMpgGetRate(mInst, 0, &velPercent);
```

```
int mcMpgGetShuttleMode(  
    MINSTANCE mInst, BOOL *on);  
  
on, rc = mc.mcMpgGetShuttleMode(  
    number mInst)  
  
<font color="green">// Check the state of MPG shuttle mode.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcMpgGetShuttleMode(mInst, &enabled);
```

```
int mcMpgGetShuttlePercent(  
    MINSTANCE mInst, int mpg, double *percent);  
  
percent, rc = mc.mcMpgGetShuttlePercent(  
    number mInst, number mpg)  
  
<font color="green">// Get the shuttle percentage for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
double percent;  
  
int rc = mcMpgGetShuttlePercentage(mInst, 0, &percent);
```

```
int mcMpgMoveCounts(  
    MINSTANCE mInst, int mpg, int deltaCounts);  
  
rc = mc.mcMpgMoveCounts(  
    number mInst, number mpg, number deltaCounts)  
  
<font color="green">// Move the MPG 0 "n" counts.</font> MINSTANCE  
mInst = 0;  
  
int n = 16;  
  
int rc = mcMpgMoveCounts(mInst, 0, n);
```

```
int mcMpgSetAccel(  
    MINSTANCE mInst, int mpg, double percentMaxAccel);  
  
rc = mc.mcMpgSetAccel(  
    number mInst, number mpg, number percentMaxAccel)  
  
<font color="green">// Set the acceleration value for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMpgSetAccel(mInst, 0, 20.5);
```

```
int mcMpgSetAxis(
```

```
    MINSTANCE mInst, int mpg, int axisId);
```

```
    rc = mc.mcMpgSetAxis(
```

```
        number mInst, number mpg, number axis)
```

```
<font color="green">// Set MPG 0 to move the X axis.</font>
```

```
    MINSTANCE mInst = 0; int rc = mcMpgSetAxis(mInst, 0, X_AXIS);
```

```
int mcMpgSetCountsPerDetent(  
    MINSTANCE mInst, int mpg, int pulses);  
  
rc = mc.mcMpgSetCountsPerDetent(  
    number mInst, number mpg, number pulses);  
  
<font color="green">// Set the number of counts per detent for MPG 0.  
</font> MINSTANCE mInst = 0;  
  
int rc = mcMpgSetCountsPerDetent(mInst, 0, 4);
```

```
int mcMpgSetEnable(  
    MINSTANCE mInst, int mpg, BOOL enabled);  
  
rc = mc.mcMpgSetEnable(  
    number mInst, number mpg number enabled)  
  
<font color="green">// Set the enabled status of MPG 0.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = MC_FALSE;  
  
int rc = mcMpgSetEnable(mInst, 0, MC_TRUE);
```

```
int mcMpgSetEncoderReg(  
    MINSTANCE mInst, int mpg, HMCREG hReg);  
  
rc = mc.mcMpgSetEncoderReg(  
    number mInst, number mpg number hReg)  
  
<font color="green">// Associate an encoder with MPG 0.</font>  
MINSTANCE mInst = 0;  
  
HMCREG hReg = 0;
```

```
int rc
```

```
rc = mcRegGetHandle(mInst, "/Sim0/Encoder0", &hReg); if (rc ==  
MERROR_NOERROR) {
```

```
    rc = mcMpgSetEncoderReg(mInst, 0, hReg); }
```

```
int mcMpgSetInc(  
    MINSTANCE mInst, int mpg, double inc);  
  
rc = mc.mcMpgSetInc(  
    number mInst, number mpg, number inc);  
  
<font color="green">// Set the increment to .001 for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMpgSetInc(mInst, 0, .001);
```

```
int mcMpgSetReversed(  
    MINSTANCE mInst, int mpg, BOOL reversed);  
  
rc = mc.mcMpgGetReversed(  
    number mInst, number mpg number reversed)  
  
<font color="green">// Get the reversed status of MPG 0.</font>  
MINSTANCE mInst = 0;  
  
BOOL reversed = MC_FALSE;  
  
int rc = mcMpgSetReversed(mInst, 0, reversed); <font color="green">//  
MC_TRUE</font>
```

```
int mcMpgSetRate(  
    MINSTANCE mInst, int mpg, double percentMaxVel);  
  
rc = mc.mcMpgSetRate(  
    number mInst, number mpg, number percentMaxVel)  
  
<font color="green">// Set the rate for MPG 0 to 25.0%</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMpgSetRate(mInst, 0, 25.0);
```

```
int mcMpgSetShuttleMode(  
    MINSTANCE mInst, BOOL on);  
  
rc = mc.mcMpgSetShuttleMode(  
    number mInst, number on);  
  
<font color="green">// Turn on MPG shuttle mode.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcMpgSetShuttleMode(mInst, MC_ON);
```

```
int mcMpgSetShuttlePercent(  
    MINSTANCE mInst, int mpg, double percent);  
  
rc = mc.mcMpgGetShuttlePercent(  
    number mInst, number mpg number percent)  
  
<font color="green">// Set the shuttle percentage for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
double percent = 200.5;  
  
int rc = mcMpgGetShuttlePercentage(mInst, 0, percent);
```



Home



Up a level



Previous



Next

Soft Limits

- [mcSoftLimitGetState](#)
- [mcSoftLimitMaxMinsClear](#)
- [mcSoftLimitSetState](#)

```
int mcSoftLimitGetState(  
    MINSTANCE mInst, int axis, double *ison);  
  
ison, rc = mc.mcSoftLimitGetState(  
    number mInst, number axis)  
  
int mInst = 0;  
  
int axis = Z_AXIS;  
  
double IsOn = 0; <font color="green">// DO_OFF to turn off.</font>  
mcSoftLimitSetState(mInst, axis, &IsOn); <font color="green">// Get the  
softlimit for the Z axis return will be 1 or 0 1==on.</font>
```

```
int mcSoftLimitSetState(  
    MINSTANCE mInst, int axis, int on);  
  
rc = mc.mcSoftLimitSetState(  
    number mInst, number axis, number on)  
  
int mInst=0;  
  
int axis = Z_AXIS;  
  
int TurnOn = MC_ON; <font color="green">// MC_OFF to turn off.</font>  
mcSoftLimitSetState(mInst, axis, TurnOn); <font color="green">// Set the  
softlimit forthe Z axis to be ON.</font>
```

```
int mcSoftLimitSetState(  
    MINSTANCE mInst, int axis, int on);  
  
rc = mc.mcSoftLimitSetState(  
    number mInst, number axis, number on)  
  
int mInst=0;  
  
int axis = Z_AXIS;  
  
int TurnOn = MC_ON; <font color="green">// MC_OFF to turn off.</font>  
mcSoftLimitSetState(mInst, axis, TurnOn); <font color="green">// Set the  
softlimit forthe Z axis to be ON.</font>
```

Spindle

- [mcSpindleCalcCSSToRPM](#)
- [mcSpindleGetAccelTime](#)
- [mcSpindleGetCommandRPM](#)
- [mcSpindleGetCurrentRange](#)
- [mcSpindleGetDecelTime](#)
- [mcSpindleGetDirection](#)
- [mcSpindleGetFeedbackRatio](#)
- [mcSpindleGetMaxRPM](#)
- [mcSpindleGetMinRPM](#)
- [mcSpindleGetMotorAccel](#)
- [mcSpindleGetMotorMaxRPM](#)
- [mcSpindleGetMotorRPM](#)
- [mcSpindleGetOverride](#)
- [mcSpindleGetOverrideEnable](#)
- [mcSpindleGetReverse](#)
- [mcSpindleGetSensorRPM](#)
- [mcSpindleGetSpeedCheckEnable](#)
- [mcSpindleGetSpeedCheckPercent](#)
- [mcSpindleGetTrueRPM](#)
- [mcSpindleSetAccelTime](#)
- [mcSpindleSetCommandRPM](#)
- [mcSpindleSetDecelTime](#)
- [mcSpindleSetDirection](#)
- [mcSpindleSetDirectionWait](#)
- [mcSpindleSetFeedbackRatio](#)
- [mcSpindleSetMaxRPM](#)
- [mcSpindleSetMinRPM](#)
- [mcSpindleSetMotorAccel](#)
- [mcSpindleSetMotorMaxRPM](#)
- [mcSpindleSetOverride](#)

- [mcSpindleSetOverrideEnable](#)
- [mcSpindleSetRange](#)
- [mcSpindleSetReverse](#)
- [mcSpindleSetSensorRPM](#)
- [mcSpindleSetSpeedCheckEnable](#)
- [mcSpindleSetSpeedCheckPercent](#)
- [mcSpindleSpeedCheck](#)

```
int mcSpindleCalcCSSToRPM(  
    MINSTANCE mInst, double DiaOfCut, BOOL Inch);  
  
rc = mc.mcSpindleCalcCSSToRPM(  
    number mInst, number DiaOfCut, number Inch);  
  
<font color="green">// make the spindle speed calc the correct RPM for a  
.550" inch cut.</font> MINSTANCE mInst = 0;  
  
int rc = mcSpindleCalcCSSToRPM(mInst, .550, TRUE);
```

```
int mcSpindleGetAccelTime(  
    MINSTANCE mInst, int Range, double *Sec);  
  
Sec, rc = mc.mcSpindleGetAccelTime(  
    number mInst, number Range)  
  
<font color="green">// Get the accel time for spindle range 2.</font>  
MINSTANCE mInst = 0;  
  
double sec = 0;  
  
int rc = mcSpindleGetAccelTime(mInst, 2, &sec);
```

```
int mcSpindleGetCommandRPM(  
    MINSTANCE mInst, double *RPM);  
  
RPM, rc = mc.mcSpindleGetCommandRPM(  
    number mInst)  
  
<font color="green">// Get the current commanded RPM</font>  
MINSTANCE mInst = 0; double rpm = 0.0;  
  
int rc = mcSpindleGetCommandRPM(mInst, &rpm);
```

```
int mcSpindleGetCurrentRange(  
    MINSTANCE mInst, int *Range);  
  
Range, rc = mc.mcSpindleGetCurrentRange(  
    number mInst)  
  
<font color="green">// Get the current spindle range.</font> MINSTANCE  
mInst = 0;  
  
int range = 0;  
  
int rc = mcSpindleGetCurrentRange(mInst, &range);
```

```
int mcSpindleGetDecelTime(  
    MINSTANCE mInst, int Range, double *Sec);  
  
Sec, rc = mc.mcSpindleGetDecelTime(  
    number mInst, number Range)  
  
<font color="green">// Get the decel. time for spindle range 3</font>  
MINSTANCE mInst = 0;  
  
double sec = 0.0;  
  
int rc = mcSpindleGetDecelTime(mInst, 3, &ec);
```

```
int mcSpindleGetDirection(  
    MINSTANCE mInst, int *dir);  
  
dir, rc = mc.mcSpindleGetDirection(  
    number mInst)  
  
<font color="green">// Get the current spindle direction.</font>  
MINSTANCE mInst = 0;  
  
int dir = MC_SPINDLE_OFF; int rc = mcSpindleGetDirection(mInst,  
&dir);
```

```
int mcSpindleGetFeedbackRatio(  
    MINSTANCE mInst, int Range, double *Ratio);  
  
Ratio, rc = mc.mcSpindleGetFeedbackRatio(  
    number mInst, number Range)  
  
<font color="green">// Get the feedback ratio for spindle range 0.</font>  
MINSTANCE mInst = 0;  
  
double ratio = 0.0;  
  
int rc = mcSpindleGetFeedbackRatio(mInst, 0, &ratio);
```

```
int mcSpindleGetMaxRPM(  
    MINSTANCE mInst, int Range, double *MaxRPM);  
  
MaxRPM, rc = mc.mcSpindleGetMaxRPM(  
    number mInst, number Range)  
  
<font color="green">// Get the max RPM for range 0.</font>  
MINSTANCE mInst = 0;  
  
double mrpm = 0.0  
  
int rc = mcSpindleGetMaxRPM(mInst, 0, &mrpm);
```

```
int mcSpindleGetMinRPM(  
    MINSTANCE mInst, int Range, double *MinRPM);  
  
MinRPM, rc = mc.mcSpindleGetMinRPM(  
    number mInst, number Range)  
  
<font color="green">// Get the min RPM for range 0.</font> MINSTANCE  
mInst = 0;  
  
double mrpm = 0.0  
  
int rc = mcSpindleGetMinRPM(mInst, 0, &mrpm);
```

```
int mcSpindleGetMotorAccel(  
    MINSTANCE mInst, int Range, double *Accel);  
  
Accel, rc = mc.mcSpindleGetMotorAccel(  
    number mInst, number Range)  
  
<font color="green">// Get the spindle acceleration for range 1.</font>  
MINSTANCE mInst = 0;  
  
double accel = 0.0;  
  
int rc = mcSpindleGetMotorAccel(mInst, 1, &accel);
```

```
int mcSpindleGetMotorMaxRPM(  
    MINSTANCE mInst, double *RPM);  
  
RPM, rc = mc.mcSpindleGetMotorMaxRPM(  
    number mInst)  
  
<font color="green">// Get the maximum spindle motor RPM.</font>  
MINSTANCE mInst = 0;  
  
double rpm = 0.0;  
  
int rc = mcSpindleGetMotorMaxRPM(mInst, &rpm);
```

```
int mcSpindleGetMotorRPM(  
    MINSTANCE mInst, double *RPM);  
  
rpm, rc = mc.mcSpindleGetMotorRPM(  
    number mInst)  
  
<font color="green">// Get the current motor RPM.</font> MINSTANCE  
mInst = 0;  
  
double rpm = 0.0;  
  
int rc = mcSpindleGetMotorRPM(mInst, &rpm);
```

```
int mcSpindleGetOverride(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mcSpindleGetOverride(  
    number mInst)  
  
<font color="green">// Get the spindle override. .5 == 50%</font>  
MINSTANCE mInst = 0;  
  
double percent = 0.0;  
  
int rc = mcSpindleGetOverride(mInst, &percent);
```

```
int mcSpindleGetOverrideEnable(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mcSpindleGetOverrideEnable(  
    number mInst)  
  
<font color="green">// Get the spindle override status.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcSpindleGetOverrideEnable(mInst, &enabled);
```

```
int mcSpindleGetReverse(  
    MINSTANCE mInst, int Range, BOOL *Reversed);  
  
BOOL Reversed, rc = mc.mcSpindleGetReverse(  
    number mInst, number Range)  
  
<font color="green">// See if the spindle is reversed in range 1</font>  
MINSTANCE mInst = 0;  
  
BOOL reversed = FALSE; int mcSpindleGetReverse(mInst, 1, &reversed);
```

```
int mcSpindleGetSensorRPM(  
    MINSTANCE mInst, double *RPM);  
  
RPM, rc = mc.mcSpindleGetSensorRPM(  
    number mInst)  
  
<font color="green">// Get the spindle sensor RPM.</font> MINSTANCE  
mInst = 0;  
  
double rpm = 0.0;  
  
int rc = mcSpindleGetSensorRPM(mInst, &rpm);
```

```
int mcSpindleGetSpeedCheckEnable(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mc.mcSpindleGetSpeedCheckEnable(  
    number mInst)  
  
<font color="green">// Check the staus of spindle speed check.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcSpindleGetSpeedCheckEnable(mInst, &enabled);
```

```
int mcSpindleGetSpeedCheckPercent(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mc.mcSpindleGetSpeedCheckPercent(  
    number mInst)  
  
<font color="green">// Get the speed check percentage.</font>  
MINSTANCE mInst = 0;  
  
double percent = 0.0;  
  
int rc = mcSpindleGetSpeedCheckPercent(mInst, &percent);
```

```
int mcSpindleGetTrueRPM(  
    MINSTANCE mInst, double *RPM);  
  
RPM, rc = mc.mcSpindleGetTrueRPM(  
    number mInst)  
  
<font color="green">// Get the true spindle RPM.</font> MINSTANCE  
mInst = 0;  
  
double rpm = 0.0;  
  
int rc = mcSpindleGetTrueRPM(mInst, &rpm);
```

```
int mcSpindleSetAccelTime(  
    MINSTANCE mInst, int Range, double Sec);  
  
rc = mc.mcSpindleSetAccelTime(  
    number mInst, number Range, number Sec);  
  
<font color="green">// Set the accel. time for spindle range 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetAccelTime(mInst, 0, 5.0);
```

```
int mcSpindleSetCommandRPM(  
    MINSTANCE mInst, double RPM);  
  
rc = mc.mcSpindleSetCommandRPM(  
    number mInst, number RPM)  
  
<font color="green">// Set the spindle RPM to 5000.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcSpindleSetCommandRPM(mInst, 5000);
```

```
int mcSpindleSetDecelTime(  
    MINSTANCE mInst, int Range, double Sec);  
  
rc = mc.mcSpindleSetDecelTime(  
    number mInst, number Range, number Sec)  
  
<font color="green">// Set the decel. time for spindle range 0.</font>  
MINSTANCE mInst = 0;  
  
in rc = mcSpindleSetDecelTime(mInst, 0, 5);
```

```
int mcSpindleSetDirection(  
    MINSTANCE mInst, int dir);  
  
rc = mc.mcSpindleSetDirection(  
    number mInst, number dir)  
  
<font color="green">// Set the spindle direction to FWD</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetDirection(mInst, MC_SPINDLE_FWD);
```

```
int mcSpindleSetDirectionWait(  
    MINSTANCE mInst, int dir);  
  
rc = mc.mcSpindleSetDirectionWait(  
    number mInst, number dir)  
  
<font color="green">// Set the spindle direction to FWD and wait</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetDirectionWait(mInst, MC_SPINDLE_FWD);
```

```
int mcSpindleSetFeedbackRatio(  
    MINSTANCE mInst, int Range, double Ratio);  
  
rc = mc.mcSpindleSetFeedbackRatio(  
    number mInst, number Range, number Ratio)  
  
<font color="green">// Set the feedback ratio for spindle range 0.</font>  
MINSTANCE mInst = 0;  
  
int mcSpindleSetFeedbackRatio(mInst, 0, 1.2); <font color="green">// 1.2 :  
1</font>
```

```
int mcSpindleSetMaxRPM(  
    MINSTANCE mInst, int Range, double RPM);  
  
rc = mc.mcSpindleSetMaxRPM(  
    number mInst, number Range, number RPM);  
  
<font color="green">// Set the max RPM for spindle range 0 to  
5000</font> MINSTANCE mInst = 0;  
  
int mcSpindleSetMaxRPM(mInst, 0, 5000);
```

```
int mcSpindleSetMinRPM(  
    MINSTANCE mInst, int Range, double RPM);  
  
rc = mc.mcSpindleSetMinRPM(  
    number mInst, number Range, number RPM);  
  
<font color="green">// Set the min RPM for spindle range 0 to 60</font>  
MINSTANCE mInst = 0;  
  
int mcSpindleSetMinRPM(mInst, 0, 60);
```

```
int mcSpindleSetMotorAccel(  
    MINSTANCE mInst, int Range, double Accel);  
  
rc = mc.mcSpindleSetMotorAccel(  
    number mInst, number Range, number Accel);  
  
<font color="green">// Set the acceleration for the spindle motor for range  
0</font> MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetMotorAccel(mInst, 0, 10000);
```

```
int mcSpindleSetMotorMaxRPM(  
    MINSTANCE mInst, double RPM);  
  
rc = mc.mcSpindleSetMotorMaxRPM(  
    number mInst, number RPM)  
  
<font color="green">// Cap the spindle motor RPM to 6000.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetMotorMaxRPM(mInst, 6000);
```

```
int mcSpindleSetOverride(  
    MINSTANCE mInst, double percent);  
  
rc = mc.mcSpindleSetOverride(  
    number mInst, number percent)  
  
<font color="green">// Set the spindle override to 50% (.5 == 50%)</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetOverride(mInst, 0.5);
```

```
int mcSpindleSetOverrideEnable(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcSpindleSetOverrideEnable(  
    number mInst, number enable)  
  
<font color="green">// Enable spindle override.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcSpindleSetOverrideEnable(mInst, TRUE);
```

```
int mcSpindleSetRange(  
    MINSTANCE mInst, int Range);  
  
rc = mc.mcSpindleSetRange(  
    number mInst, number Range)  
  
<font color="green">// Set the current spindle range to the first range (0).  
</font> MINSTANCE mInst = 0;  
  
mcSpindleSetRange(mInst, 0);
```

```
int mcSpindleSetReverse(  
    MINSTANCE mInst, int Range, BOOL Reversed);  
  
rc = mc.mcSpindleSetReverse(  
    number mInst, number Range, number Reversed)  
  
<font color="green">// Set spindle range 1 as being reversed.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetReverse(mInst, 1, TRUE);
```

```
int mcSpindleSetSensorRPM(  
    MINSTANCE mInst, double RPM);  
  
rc = mc.mcSpindleSetSensorRPM(  
    number mInst, number RPM)  
  
<font color="green">// Set the spindle sensor RPM to 1000</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetSensorRPM(mInst, 1000);
```

```
int mcSpindleSetSpeedCheckEnable(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcSpindleSetSpeedCheckEnable(  
    number mInst, number enable)  
  
<font color="green">// Enable spindle speed checking.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetSpeedCheckEnable(mInst, TRUE);
```

```
int mcSpindleSetSpeedCheckPercent(  
    MINSTANCE mInst, double percent);  
  
rc = mc.mcSpindleSetSpeedCheckPercent(  
    number mInst, number percent)  
  
<font color="green">// Set the speed check percentage to 5%. 100 == 100%  
</font> MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetSpeedCheckPercent(mInst, 5);
```

```
int mcSpindleSpeedCheck(  
    MINSTANCE mInst, int *SpeedOk);  
  
SpeedOk, rc = mc.mcSpindleSpeedCheck(  
    number mInst)  
  
<font color="green">// Check to see if the spidnle speed is in the allowed  
range.</font> MINSTANCE mInst = 0; BOOL speedOk;  
  
int rc = mcSpindleSpeedCheck(mInst, &speedOk);
```

Scripting

- [mcScriptDebug](#)
- [mcScriptEngineRegister](#)
- [mcScriptExecute](#)
- [mcScriptExecuteIfExists](#)
- [mcScriptExecutePrivate](#)
- [mcScriptGetExtensions](#)

```
int mcScriptDebug(  
    MINSTANCE mInst, const char *filename);
```

N/A

```
<font color="green">// Debug the M6 macro</font> MINSTANCE mInst =  
0; int rc = mcScriptDebug(mInst, "m6.mcs");
```

```
int mcScriptEngineRegister(
```

```
    MINSTANCE mInst, HMCPLUG pluginid, const char *EngineName,  
    const char *EngineDesc, const char *FileExtensions);
```

N/A

```
<font color="green">// Register the LUA script engine.</font>  
MINSTANCE mInst = 0;
```

```
HMCPLUG id; <font color="green">// From mcPluginInit()</font>; int rc  
= mcScriptEngineRegister(mInst, id, "wxLua", "wxLua script engine",  
"mcs|lua|wx.lua");
```

```
int mcScriptExecute(  
    MINSTANCE mInst, const char *filename, BOOL async);  
  
rc = mc.mcScriptExecute(  
    number mInst, string filename, number async)  
  
<font color="green">// Execute the M6 script and wait on it to complete.  
</font> MINSTANCE mInst = 0;  
  
rc = mcScriptExecute(mInst, "m6.mcs", FALSE);
```

```
int mcScriptExecuteIfExists(  
    MINSTANCE mInst, const char *filename, BOOL async);  
  
rc = mc.mcScriptExecuteIfExists(  
    number mInst, string filename, number async)  
  
<font color="green">// Execute the M6 script and wait on it to complete.  
</font> MINSTANCE mInst = 0;  
  
rc = mcScriptExecute(mInst, "m6.mcs", FALSE);
```

```
int mcScriptExecutePrivate(  
    MINSTANCE mInst, const char *filename, BOOL async);  
  
rc = mc.mcScriptExecutePrivate(  
    number mInst, string filename, BOOL async)  
  
<font color="green">// Execute myScript.mcs asynchronously in it's own  
private environment.</font> MINSTANCE mInst = 0;  
  
int rc = mcScriptExecutePrivate(mInst, "myScript.mcs", TRUE);
```

```
int mcScriptGetExtensions(MINSTANCE mInst, char *buf, long bufsize);
```

N/A

```
<font color="green">// Retrieve all of the registered script extensions.  
</font> MINSTANCE mInst = 0; mcScriptGetExtensions(MINSTANCE  
mInst, char *buf, long bufsize);
```

Status Messages

- [mcCtlGetLastError](#)
- [mcCtlGetLastLogMsg](#)
- [mcCtlLog](#)
- [mcCtlSetLastError](#)
- [mcCtlSetLogging](#)

```
int mcCntrlGetLastError(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCntrlGetLastError(  
    MINSTANCE mInst)  
  
int mInst=0;  
  
char error[128]; mcCntrlGetLastError(mInst, error, 128); <font  
color="green">// Receives the last error messge for controller instance 0.  
</font>
```

```
int mcCntrlGetLastLogMsg(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCntrlGetLastLogMsg(  
    number mInst)  
  
int mInst=0;  
  
char msg[128]; mcCntrlGetLastLogMsg(mInst, msg, 128); <font  
color="green">// Receives the last log messge for controller instance 0.  
</font>
```

```
int mcCntlLog(  
    MINSTANCE mInst, const char *message, const char *file, int line);  
  
rc = mc.mcCntlLog(  
    number mInst, string message, string file, number line)  
  
<font color="green">// Load G code from a string.</font> MINSTANCE  
mInst = 0; BOOL test = TRUE;  
  
int rc;  
  
if (test == TRUE) {  
  
    <font color="green">// File and line info in the log.</font>  
    mcCntlLog(mInst, "test == TRUE!", __FILE__, __LINE__); } else {  
  
    <font color="green">// No file and line info in the log.</font>  
    mcCntlLog(mInst, "test == FALSE!", NULL, 0); }
```

```
int mcCtlSetLastError(  
    MINSTANCE mInst, const char *emsg);  
  
rc = mc.mcCtlSetLastError(  
    number mInst, string emsg)  
  
MINSTANCE mInst = 0;  
  
char *erbuf = "Tool no longer in the spindle"; <font color="green">// Send  
an error message for controller 0.</font> int rc = mcCtlSetLastError(mInst,  
erbuf);
```

```
int mcCtlSetLogging(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcCtlSetLogging(  
    number mInst, number enable)  
  
<font color="green">// Enable logging.</font> MINSTANCE mInst = 0;  
  
int rc = mcCtlSetLogging(mInst, enable);
```



Home



Up a level



Previous



Next

License

- [mcCtlCheckLicenseFeature](#)
- [mcCtlGetLicenseData](#)
- [mcCtlGetLicenseDataLen](#)
- [mcCtlGetLicenseModules](#)

```
int mcCtlCheckLicenseFeature(  
    MINSTANCE mInst, const char *licFile, const char *requirement, const  
    char *feature);  
  
rc = mcCtlCheckLicenseFeature(  
    number mInst, string licFile, string requirement, string feature);  
  
<font color="green">// Check the DarwinLic.dat file for feature  
M4_DARWIN</font> MINSTANCE mInst = 0;  
  
int rc = mcCtlCheckLicenseFeature(mInst, "DarwinLic.dat",  
    "NF_DARWIN", "M4_DARWIN"); if (rc != MERROR_NOERROR) {  
  
    <font color="green">// Feature found!!!</font> return; }
```

```
int mcCtlGetLicenseData(  
    MINSTANCE mInst, int index, char *buf, long bufSize);  
  
buf, rc = mc.mcCtlGetLicenseData(  
    number mInst, number index)  
  
<font color="green">// Get the license data at index 1.</font>  
MINSTANCE mInst = 0;  
  
long len;  
  
char *data;  
  
long dataLen = 0;  
  
int index = 1;  
  
int rc = mcCtlGetLicenseDataLen(mInst, index, &dataLen); if (rc ==  
MERROR_NOERROR && dataLen > 0) {  
  
    data = (char *)malloc(dataLen + 1); memset(data, 0, dataLen + 1); rc =  
<font color="blue">mcCtlGetLicenseData(mInst, index, data, dataLen)  
</font>; if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> }  
  
}
```

```
int mcCtlGetLicenseDataLen(  
    MINSTANCE mInst, int index, long *bufSize);  
  
bufSize, rc = mc.mcCtlGetLicenseData(  
    number mInst, number index)  
  
<font color="green">// Get the license data at index 1.</font>  
MINSTANCE mInst = 0;  
  
long len;  
  
char *data;  
  
long dataLen = 0;  
  
int index = 1;  
  
int rc = <font color="blue">mcCtlGetLicenseDataLen(mInst, index,  
&dataLen)</font>; if (rc == MERROR_NOERROR && dataLen > 0) {  
  
    data = (char *)malloc(dataLen + 1); memset(data, 0, dataLen + 1); rc =  
    mcCtlGetLicenseData(mInst, index, data, dataLen); if (rc ==  
    MERROR_NOERROR) {  
  
        <font color="green">// Success!</font> }  
  
    }  
}
```

```
int mcCtlGetLicenseModules(
```

```
    MINSTANCE mInst, unsigned long long *modules)
```

N/A

// Retrieve the licensed modules.

```
MINSTANCE mInst = 0;
```

```
unsigned long long modules = 0; int rc = mcCtlGetLicenseModules(mInst,  
&modules);
```



Home



Previous



Next

Alphabetical API Reference

- [mcAxisDeref](#)
- [mcAxisDerefAll](#)
- [mcAxisE2nable](#)
- [mcAxisGetHomeDir](#)
- [mcAxisGetHomeInPlace](#)
- [mcAxisGetHomeOffset](#)
- [mcAxisGetHomeOrder](#)
- [mcAxisGetHomeSpeed](#)
- [mcAxisGetInfoStruct](#)
- [mcAxisGetMachinePos](#)
- [mcAxisGetMotorId](#)
- [mcAxisGetOverrideAxis](#)
- [mcAxisGetPos](#)
- [mcAxisGetProbePos](#)
- [mcAxisGetProbePosAll](#)
- [mcAxisGetScale](#)
- [mcAxisGetSoftlimitEnable](#)

- [mcAxisGetSoftlimitMax](#)
- [mcAxisGetSoftlimitMin](#)
- [mcAxisGetSpindle](#)
- [mcAxisGetVel](#)
- [mcAxisHome](#)
- [mcAxisHomeAll](#)
- [mcAxisHomeComplete](#)
- [mcAxisHomeCompleteWithStatus](#)
- [mcAxisIsEnabled](#)
- [mcAxisIsHomed](#)
- [mcAxisIsHoming](#)
- [mcAxisIsStill](#)
- [mcAxisMapMotor](#)
- [mcAxisRegister](#)
- [mcAxisRemoveOverrideAxis](#)
- [mcAxisRemoveOverrideAxisSync](#)
- [mcAxisSetHomeDir](#)
- [mcAxisSetHomeInPlace](#)
- [mcAxisSetHomeOffset](#)
- [mcAxisSetHomeOrder](#)

- [mcAxisSetHomeSpeed](#)
- [mcAxisSetInfoStruct](#)
- [mcAxisSetMachinePos](#)
- [mcAxisSetOverrideAxis](#)
- [mcAxisSetPos](#)
- [mcAxisSetSoftlimitEnable](#)
- [mcAxisSetSoftlimitMax](#)
- [mcAxisSetSoftlimitMin](#)
- [mcAxisSetSpindle](#)
- [mcAxisSetVel](#)
- [mcAxisUnmapMotor](#)
- [mcAxisUnmapMotors](#)
- [mcAxisUnregister](#)
- [mcCtlCheckLicenseFeature](#)
- [mcCtlCleanup](#)
- [mcCtlCloseGCodeFile](#)
- [mcCtlCompileScripts](#)
- [mcCtlConfigStart](#)
- [mcCtlConfigStop](#)
- [mcCtlCreateLocalVars](#)

- [mcCtlCycleStart](#)
- [mcCtlCycleStop](#)
- [mcCtlDryRunToLine](#)
- [mcCtlEStop](#)
- [mcCtlEnable](#)
- [mcCtlFeedHold](#)
- [mcCtlFeedHoldState](#)
- [mcCtlGcodeExecute](#)
- [mcCtlGcodeExecuteWait](#)
- [mcCtlGcodeInterpGetData](#)
- [mcCtlGcodeInterpGetPos](#)
- [mcCtlGetBlockDelete](#)
- [mcCtlGetBuild](#)
- [mcCtlGetComputerID](#)
- [mcCtlGetCoolantDelay](#)
- [mcCtlGetCwd](#)
- [mcCtlGetDiaMode](#)
- [mcCtlGetDistToGo](#)
- [mcCtlGetEnableFRO](#)
- [mcCtlGetFRO](#)

- [mcCtlGetGcodeFileName](#)
- [mcCtlGetGcodeLine](#)
- [mcCtlGetGcodeLineCount](#)
- [mcCtlGetGcodeLineNbr](#)
- [mcCtlGetInstanceHandle](#)
- [mcCtlGetLastError](#)
- [mcCtlGetLastLogMsg](#)
- [mcCtlGetLicenseData](#)
- [mcCtlGetLicenseDataLen](#)
- [mcCtlGetLicenseModules](#)
- [mcCtlGetLocalComment](#)
- [mcCtlGetLocalVar](#)
- [mcCtlGetLogging](#)
- [mcCtlGetMachDir](#)
- [mcCtlGetMistDelay](#)
- [mcCtlGetModalGroup](#)
- [mcCtlGetMode](#)
- [mcCtlGetOffset](#)
- [mcCtlGetOptionalStop](#)
- [mcCtlGetPoundVar](#)

- [mcCtlGetRRO](#)
- [mcCtlGetRunTime](#)
- [mcCtlGetSingleBlock](#)
- [mcCtlGetState](#)
- [mcCtlGetStateName](#)
- [mcCtlGetStats](#)
- [mcCtlGetToolOffset](#)
- [mcCtlGetUnitsCurrent](#)
- [mcCtlGetUnitsDefault](#)
- [mcCtlGetValue](#)
- [mcCtlGetVersion](#)
- [mcCtlGotoZero](#)
- [mcCtlInit](#)
- [mcCtlIsInCycle](#)
- [mcCtlIsStill](#)
- [mcCtlLoadGcodeFile](#)
- [mcCtlLoadGcodeString](#)
- [mcCtlLog](#)
- [mcCtlMachineStateClear](#)
- [mcCtlMachineStatePop](#)

- [mcCtlMachineStatePush](#)
- [mcCtlMacroAlarm](#)
- [mcCtlMacroStop](#)
- [mcCtlMdiExecute](#)
- [mcCtlProbeFileClose](#)
- [mcCtlProbeFileOpen](#)
- [mcCtlProbeGetStrikeStatus](#)
- [mcCtlReset](#)
- [mcCtlRewindFile](#)
- [mcCtlSetBlockDelete](#)
- [mcCtlSetCoolantDelay](#)
- [mcCtlSetDiaMode](#)
- [mcCtlSetEnableFRO](#)
- [mcCtlSetFRO](#)
- [mcCtlSetGcodeLineNbr](#)
- [mcCtlSetLastError](#)
- [mcCtlSetLogging](#)
- [mcCtlSetMistDelay](#)
- [mcCtlSetMode](#)
- [mcCtlSetOptionalStop](#)

- [mcCtlSetPoundVar](#)
- [mcCtlSetRRO](#)
- [mcCtlSetResetCodes](#)
- [mcCtlSetSingleBlock](#)
- [mcCtlSetStats](#)
- [mcCtlSetValue](#)
- [mcCtlStartMotionDev](#)
- [mcCtlStopMotionDev](#)
- [mcCtlToolChangeManual](#)
- [mcCtlWaitOnCycleStart](#)
- [mcDeviceGetHandle](#)
- [mcDeviceGetInfo](#)
- [mcDeviceGetInfoStruct](#)
- [mcDeviceGetNextHandle](#)
- [mcDeviceRegister](#)
- [mcFileHoldAquire](#)
- [mcFileHoldReason](#)
- [mcFileHoldRelease](#)
- [mcFixtureLoadFile](#)
- [mcFixtureSaveFile](#)

- [mcGuiGetWindowHandle](#)
- [mcGuiSetCallback](#)
- [mcGuiSetFocus](#)
- [mcGuiSetWindowHandle](#)
- [mcIoGetHandle](#)
- [mcIoGetInfoStruct](#)
- [mcIoGetNextHandle](#)
- [mcIoGetState](#)
- [mcIoGetType](#)
- [mcIoGetUserData](#)
- [mcIoIsEnabled](#)
- [mcIoRegister](#)
- [mcIoSetDesc](#)
- [mcIoSetName](#)
- [mcIoSetState](#)
- [mcIoSetType](#)
- [mcIoSetUserData](#)
- [mcIoSyncSignal](#)
- [mcIoUnregister](#)
- [mcJogAbsStart](#)

- [mcJogAbsStop](#)
- [mcJogGetAccel](#)
- [mcJogGetFeedRate](#)
- [mcJogGetFollowMode](#)
- [mcJogGetInc](#)
- [mcJogGetRate](#)
- [mcJogGetTraceEnable](#)
- [mcJogGetType](#)
- [mcJoggetUnitsMode](#)
- [mcJogGetVelocity](#)
- [mcJogIncStart](#)
- [mcJogIncStop](#)
- [mcJogIsJogging](#)
- [mcJogIsStopping](#)
- [mcJogSetAccel](#)
- [mcJogSetFeedRate](#)
- [mcJogSetFollowMode](#)
- [mcJogSetInc](#)
- [mcJogSetRate](#)
- [mcJogSetTraceEnable](#)

- [mcJogSetType](#)
- [mcJogSetUnitsMode](#)
- [mcJogVelocityStart](#)
- [mcJogVelocityStop](#)
- [mcMotionClearPlanner](#)
- [mcMotionCyclePlanner](#)
- [mcMotionCyclePlannerEx](#)
- [mcMotionGetAbsPos](#)
- [mcMotionGetAbsPosFract](#)
- [mcMotionGetBacklashAbs](#)
- [mcMotionGetBacklashInc](#)
- [mcMotionGetIncPos](#)
- [mcMotionGetMoveID](#)
- [mcMotionGetPos](#)
- [mcMotionGetProbeParams](#)
- [mcMotionGetRigidTapParams](#)
- [mcMotionGetSyncOutput](#)
- [mcMotionGetThreadParams](#)
- [mcMotionGetThreadingRate](#)
- [mcMotionGetVel](#)

- [mcMotionSetCycleTime](#)
- [mcMotionSetMoveID](#)
- [mcMotionSetPos](#)
- [mcMotionSetProbeComplete](#)
- [mcMotionSetProbePos](#)
- [mcMotionSetStill](#)
- [mcMotionSetThreadingRate](#)
- [mcMotionSetVel](#)
- [mcMotionSync](#)
- [mcMotionThreadComplete](#)
- [mcMotorGetAxis](#)
- [mcMotorGetCountsPerUnit](#)
- [mcMotorGetInfoStruct](#)
- [mcMotorGetPos](#)
- [mcMotorGetVel](#)
- [mcMotorIsHomed](#)
- [mcMotorIsStill](#)
- [mcMotorMapGetDefinition](#)
- [mcMotorMapGetLength](#)
- [mcMotorMapGetNPoints](#)

- [mcMotorMapGetPoint](#)
- [mcMotorMapGetPointCount](#)
- [mcMotorMapGetStart](#)
- [mcMotorMapSetDefinition](#)
- [mcMotorMapSetLength](#)
- [mcMotorMapSetNPoints](#)
- [mcMotorMapSetPoint](#)
- [mcMotorMapSetPointCount](#)
- [mcMotorMapSetStart](#)
- [mcMotorRegister](#)
- [mcMotorSetCountsPerUnit](#)
- [mcMotorSetHomePos](#)
- [mcMotorSetInfoStruct](#)
- [mcMotorUnregister](#)
- [mcMpgGetAccel](#)
- [mcMpgGetAxis](#)
- [mcMpgGetCountsPerDetent](#)
- [mcMpgGetEnable](#)
- [mcMpgGetEncoderReg](#)
- [mcMpgGetInc](#)

- [mcMpgGetReversed](#)
- [mcMpgGetRate](#)
- [mcMpgGetShuttleMode](#)
- [mcMpgGetShuttlePercent](#)
- [mcMpgMoveCounts](#)
- [mcMpgSetAccel](#)
- [mcMpgSetAxis](#)
- [mcMpgSetCountsPerDetent](#)
- [mcMpgSetEnable](#)
- [mcMpgSetEncoderReg](#)
- [mcMpgSetInc](#)
- [mcMpgSetReversed](#)
- [mcMpgSetRate](#)
- [mcMpgSetShuttleMode](#)
- [mcMpgSetShuttlePercent](#)
- [mcPluginConfigure](#)
- [mcPluginCoreLoad](#)
- [mcPluginCoreUnload](#)
- [mcPluginDiagnostic](#)
- [mcPluginEnable](#)

- [mcPluginGetEnabled](#)
- [mcPluginGetInfoStruct](#)
- [mcPluginGetLicenseFeature](#)
- [mcPluginGetNextHandle](#)
- [mcPluginGetValid](#)
- [mcPluginInstall](#)
- [mcPluginRegister](#)
- [mcPluginRemove](#)
- [mcProfileDeleteKey](#)
- [mcProfileDeleteSection](#)
- [mcProfileExists](#)
- [mcProfileFlush](#)
- [mcProfileGetDouble](#)
- [mcProfileGetInt](#)
- [mcProfileGetName](#)
- [mcProfileGetString](#)
- [mcProfileReload](#)
- [mcProfileSave](#)
- [mcProfileWriteDouble](#)
- [mcProfileWriteInt](#)

- [mcProfileWriteString](#)
- [mcRegGetCommand](#)
- [mcRegGetHandle](#)
- [mcRegGetInfo](#)
- [mcRegGetInfoStruct](#)
- [mcRegGetNextHandle](#)
- [mcRegGetUserData](#)
- [mcRegGetValue](#)
- [mcRegGetValueLong](#)
- [mcRegGetValueString](#)
- [mcRegGetValueStringClear](#)
- [mcRegRegister](#)
- [mcRegSendCommand](#)
- [mcRegSetDesc](#)
- [mcRegSetName](#)
- [mcRegSetResponse](#)
- [mcRegSetType](#)
- [mcRegSetUserData](#)
- [mcRegSetValue](#)
- [mcRegSetValueLong](#)

- [mcRegSetValueString](#)
- [mcRegUnregister](#)
- [mcScriptDebug](#)
- [mcScriptEngineRegister](#)
- [mcScriptExecute](#)
- [mcScriptExecuteIfExists](#)
- [mcScriptExecutePrivate](#)
- [mcScriptGetExtensions](#)
- [mcSignalEnable](#)
- [mcSignalGetHandle](#)
- [mcSignalGetInfo](#)
- [mcSignalGetInfoStruct](#)
- [mcSignalGetNextHandle](#)
- [mcSignalGetState](#)
- [mcSignalHandleWait](#)
- [mcSignalMap](#)
- [mcSignalSetActiveLow](#)
- [mcSignalSetState](#)
- [mcSignalUnmap](#)
- [mcSignalWait](#)

- [mcSoftLimitGetState](#)
- [mcSoftLimitMaxMinsClear](#)
- [mcSoftLimitSetState](#)
- [mcSpindleCalcCSSToRPM](#)
- [mcSpindleGetAccelTime](#)
- [mcSpindleGetCommandRPM](#)
- [mcSpindleGetCurrentRange](#)
- [mcSpindleGetDecelTime](#)
- [mcSpindleGetDirection](#)
- [mcSpindleGetFeedbackRatio](#)
- [mcSpindleGetMaxRPM](#)
- [mcSpindleGetMinRPM](#)
- [mcSpindleGetMotorAccel](#)
- [mcSpindleGetMotorMaxRPM](#)
- [mcSpindleGetMotorRPM](#)
- [mcSpindleGetOverride](#)
- [mcSpindleGetOverrideEnable](#)
- [mcSpindleGetReverse](#)
- [mcSpindleGetSensorRPM](#)
- [mcSpindleGetSpeedCheckEnable](#)

- [mcSpindleGetSpeedCheckPercent](#)
- [mcSpindleGetTrueRPM](#)
- [mcSpindleSetAccelTime](#)
- [mcSpindleSetCommandRPM](#)
- [mcSpindleSetDecelTime](#)
- [mcSpindleSetDirection](#)
- [mcSpindleSetDirectionWait](#)
- [mcSpindleSetFeedbackRatio](#)
- [mcSpindleSetMaxRPM](#)
- [mcSpindleSetMinRPM](#)
- [mcSpindleSetMotorAccel](#)
- [mcSpindleSetMotorMaxRPM](#)
- [mcSpindleSetOverride](#)
- [mcSpindleSetOverrideEnable](#)
- [mcSpindleSetRange](#)
- [mcSpindleSetReverse](#)
- [mcSpindleSetSensorRPM](#)
- [mcSpindleSetSpeedCheckEnable](#)
- [mcSpindleSetSpeedCheckPercent](#)
- [mcSpindleSpeedCheck](#)

- [mcToolGetCurrent](#)
- [mcToolGetData](#)
- [mcToolGetDataExDbl](#)
- [mcToolGetDataExInt](#)
- [mcToolGetDataExStr](#)
- [mcToolGetDesc](#)
- [mcToolGetSelected](#)
- [mcToolLoadFile](#)
- [mcToolPathCreate](#)
- [mcToolPathDelete](#)
- [mcToolPathGenerate](#)
- [mcToolPathGenerateAbort](#)
- [mcToolPathGeneratedPercent](#)
- [mcToolPathGetAAxisPosition](#)
- [mcToolPathGetARotationAxis](#)
- [mcToolPathGetAxisColor](#)
- [mcToolPathGetBackColor](#)
- [mcToolPathGetDrawLimits](#)
- [mcToolPathGetExecution](#)
- [mcToolPathGetFollowMode](#)

- [mcToolPathGetGenerating](#)
- [mcToolPathGetLeftMouseDn](#)
- [mcToolPathGetLeftMouseUp](#)
- [mcToolPathGetPathColor](#)
- [mcToolPathIsSignalMouseClicks](#)
- [mcToolPathSetAAxisPosition](#)
- [mcToolPathSetARotationAxis](#)
- [mcToolPathSetAxisColor](#)
- [mcToolPathSetBackColor](#)
- [mcToolPathSetDrawLimits](#)
- [mcToolPathSetFollowMode](#)
- [mcToolPathSetPathColor](#)
- [mcToolPathSetSignalMouseClicks](#)
- [mcToolPathSetView](#)
- [mcToolPathSetZoom](#)
- [mcToolPathUpdate](#)
- [mcToolSaveFile](#)
- [mcToolSetCurrent](#)
- [mcToolSetData](#)
- [mcToolSetDataExDbl](#)

- [mcToolSetDataExInt](#)
- [mcToolSetDataExStr](#)
- [mcToolSetDesc](#)

```
int mcAxisDeref(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisDeref(  
    number mInst, number axisId)  
  
int mInst = 0;  
  
<font color="green">// Set AXIS0 to deref (could have used X_AXIS).  
</font> mcAxisDeref(mInst, AXIS0);
```

```
int mcAxisDerefAll(  
    MINSTANCE mInst);  
  
rc = mc.mcAxisDerefAll(  
    number mInst)  
  
int mInst = 0;  
  
<font color="green">// Set all axis to deref.</font>  
mcAxisDerefAll(mInst);
```

```
int mcAxisEnable(  
    MINSTANCE mInst, int axisId, BOOL enabled);  
  
rc = mc.mcAxisEnable(  
    number mInst, number axisId, number enabled);  
  
int mInst = 0;  
  
<font color="green">// Set Y_AXIS to be disabled.</font>  
mcAxisEnable(mInst, Y_AXIS, false);
```

```
int mcAxisGetHomeDir(  
    MINSTANCE mInst, int axisId, int *dir);  
  
dir, rc = mc.mcAxisGetHomeDir(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int dir = 0;  
  
<font color="green">// Get Home offset of the X axis.</font>  
mcAxisGetHomeDir(mInst, X_AXIS, &dir);
```

```
int mcAxisGetHomeInPlace(  
    MINSTANCE mInst, int axisId, BOOL *homeInPlace);  
  
homeInPlace, rc = mc.mcAxisGetHomeInPlace(  
    number mInst, number axisId)  
  
<font color="green">// Get the Home In Place flag for the X axis.</font>  
MINSTANCE mInst = 0;  
  
BOOL hip = FALSE;  
  
mcAxisGetHomeInPlace(mInst, X_AXIS, &hip);
```

```
int mcAxisGetHomeOffset(  
    MINSTANCE mInst, int axisId, double *offset);  
  
offset, rc = mc.mcAxisGetHomeOffset(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int offset = 0;  
  
<font color="green">// Get Home offset of the X axis.</font>  
mcAxisGetHomeOrder(mInst, X_AXIS, &offset);
```

```
int mcAxisGetHomeOrder(  
    MINSTANCE mInst, int axisId, int *order);  
  
order, rc = mc.mcAxisGetHomeOrder(  
    number mInst, number axisId)  
  
int mInst=0;  
  
double XAxisPos = 0;  
  
int order = 0;  
  
<font color="green">// Get Home order of the X axis.</font>  
mcAxisGetHomeOrder(mInst, X_AXIS, &order);
```

```
int mcAxisGetHomeSpeed(  
    MINSTANCE mInst, int axisId, double *percent);  
  
percent, rc = mc.mcAxisGetHomeSpeed(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int rc = 0;  
  
double XAxisPos = 0; double percent = 0; double maxVel = 0  
  
double homeVel = 0;  
  
<font color="green">// Get home speed percentage of the X axis.</font>  
mcAxisGetHomeSpeed(mInst, X_AXIS, &percent); mcAxisGetVel(mInst,  
X_AXIS, &maxVel); homeVel = maxVel * percent;
```

```
int mcAxisGetInfoStruct(  
    MINSTANCE mInst,  
    int axisId,  
    axisinfo_t *ainf);  
  
ainf, rc = mc.mcAxisGetInfoStruct(number mInst, number axisId)  
  
struct axisinfo {  
  
    BOOL OutOfBandAxis; // Is this an out of band axis?  
  
    BOOL IsStill; // Set high when the axis is not moving BOOL Jogging; //  
    Used to tell to jog...  
  
    BOOL Homing; // Used to tell the state of the home operation.  
  
    int Id; // Axis Id  
  
    BOOL IsSpindle; // Does this axis control a spindle?  
  
    BOOL Enabled; // Is axis enabled?  
  
    BOOL SoftlimitEnabled; // Softlimits enabled?  
  
    double SoftMaxLimit; // Count for the max travel.  
  
    double SoftMinLimit; // Count for the m.  
  
    BOOL VelocityMode; // Used to make the axis move at a fixed speed  
    BOOL BufferJog; // Buffer all jogs?  
  
    double Pos; // Position in user units.  
  
    double Mpos; // Machine position in user units.  
  
    int HomeOrder; // The order in which to home the axis.
```

```
int HomeDir; // The direction the axis homes.

double HomeOffset; // The offset from the the limits switch.

double HomeSpeedPercent;// The percentage of the max velocity at
which to home.

BOOL SoftlimitUsed; // Use Softlimits?

BOOL HomeInPlace; // Zero the axis in place when Refed?

int MotorId[MC_MAX_AXIS_MOTORS]; //child motor ID array.

};

typedef struct axisinfo axisinfo_t;

int mInst = 0;

axisinfo_t ainf;

<font color="green">// Get Y_AXIS info structure.</font>
mcAxisGetInfoStruct(mInst, Y_AXIS, &ainf);
```

```
int mcAxisGetMachinePos(  
    MINSTANCE mInst, int axisId, double *val);  
  
val, rc = mc.mcAxisGetMachinePos(  
    number mInst, number axisId)  
  
int mInst=0;  
  
double XAxisMachinePos = 0; int AxisNumber = X_AXIS;  
  
mcAxisGetMachinePos(mInst, AxisNumber, &XAxisMachinePos); <font  
color="green">// Get the position of the X axis in Machine Pos.</font>
```

```
int mcAxisGetMotorId(  
    MINSTANCE mInst, int axis, int childId, int *motorId);  
  
motorId, rc = mc.mcAxisGetMotorId(  
    number mInst, number axis, number childId)  
  
int mInst=0;  
  
int motorIds[]={-1,-1,-1,-1,-1}  
  
int id;  
  
<font color="green">// Get all the motor ID's for the X axis up to 5 motors.  
</font> for(int i = 0; i < 5; i++){  
  
    if (mcAxisGetMotorId(mInst, X_AXIS, i, &id) ==  
  
        MERROR_NOERROR No Error.) {  
  
        motorIds[i] = id; }  
  
    }  
}
```

```
int mcAxisGetOverrideAxis(  
    MINSTANCE mInst, int axis, int *axis1, int *axis2, int *axis3, int  
    *axis4);  
  
axis1, axis2, axis3, axis4, rc = mc.mcAxisGetOverrideAxis(  
    number mInst, number axis)  
  
int mInst = 0;  
  
int ora[4];  
  
<font color="green">// Get the override axis for the Z axis.</font>  
mcAxisGetOverrideAxis(mInst, ZAXIS, &ora[0], &ora[1], &ora[2],  
    &ora[3]);
```

```
int mcAxisGetPos(  
    MINSTANCE mInst, int axisId, double *val);  
  
val, rc = mc.mcAxisGetPos(  
    number mInst, number axisId)  
  
int mInst=0;  
  
double XAxisPos = 0;  
  
int AxisNumber = X_AXIS; <font color="green">// Get the position of the  
X axis.</font> mcAxisGetPos(mInst, AxisNumber, &XAxisPos);
```

```
int mcAxisGetProbePos(  
    MINSTANCE mInst, int axisId, BOOL machinePos, double *val);  
  
val, rc = mc.mcAxisGetProbePos(  
    number mInst, number axisId, number machinePos)  
  
<font color="green">// Get the last probe strike position for the X  
axis</font> MINSTANCE mInst = 0;  
  
double xProbePos = 0;  
  
mcAxisGetProbePos(mInst, X_AXIS, FALSE, &xProbPos);
```

```
int mcAxisGetProbePosAll(  
    MINSTANCE mInst, BOOL machinePos, double *x, double *y, double  
    *z, double *a, double *b, double *c);  
  
x, y, z, a, b, c, rc = mc.mcAxisGetProbePosAll(  
    number mInst, number machinePos)  
  
<font color="green">// Get the last probe strike position for the X, Y, and Z  
axes.</font> MINSTANCE mInst = 0;  
  
double x, y, z;  
  
int rc = mcAxisGetProbePosAll(mInst, FALSE, &x, &y, &z, NULL,  
NULL, NULL); if (rc == MERROR_NOERROR) {  
  
    // Process probe positions...  
  
}
```

```
int mcAxisGetScale(  
    MINSTANCE mInst, int axisId, double *scaleVal);  
  
scaleVal, rc = mc.mcAxisGetProbePos(  
    number mInst, number axisId)  
  
<font color="green">// Get the scale value for the X axis</font>  
MINSTANCE mInst = 0; double scale = 0;  
  
mcAxisGetScale(mInst, X_AXIS, &scale);
```

```
int mcAxisGetSoftlimitEnable(  
    MINSTANCE mInst, int axisId, int *enable);  
  
enable, rc = mc.mcAxisGetSoftlimitEnable(  
    number mInst, number axisId)  
  
<font color="green">// Get master soft limit enable state for axis 0.</font>  
int mInst = 0;  
  
int enable = 0;  
  
mcAxisGetSoftlimitEnable(mInst, 0, &enable);
```

```
int mcAxisGetSoftlimitMax(  
    MINSTANCE mInst, int axisId, double *max);  
  
max, rc = mc.mcAxisGetSoftlimitMin(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int axis = Y_AXIS; double min = 0;  
  
mcAxisGetSoftlimitMax(mInst, axis, &min);<font color="green">// Get the  
max distance of the softlimit for the Y axis.</font>
```

```
int mcAxisGetSoftlimitMin(  
    MINSTANCE mInst, int axisId, double *min);  
  
min, rc = mc.mcAxisGetSoftlimitMin(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int axis = Y_AXIS; double min = 0;  
  
mcAxisGetSoftlimitMin(mInst, axis, &min);<font color="green">// Get the  
min distance of the softlimit for the Y axis.</font>
```

```
int mcAxisGetSpindle(  
    MINSTANCE mInst, int axisId, bool *spindle);  
  
spindle, rc = mcAxisGetSpindle(  
    number mInst, number axisId)  
  
<font color="green">// Find the spindle axis</font> int mInst = 0;  
int axisId;  
  
bool isSpindle = false; for (axisId = MC_MAX_COORD_AXES; axisId <  
MC_MAX_AXES; axisId++) {  
  
    mcAxisGetSpindle(mInst, axisId, &isSpindle); if (isSpindle) {  
  
        <font color="green">// Spindle found!</font> break; }  
  
    }  
}
```

```
int mcAxisGetVel(  
    MINSTANCE mInst, int axisId, double *velocity);  
  
velocity, rc = mc.mcAxisGetVel(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double CurrentVel = 0; mcAxisGetVel(mInst, axis, &CurrentVel); <font  
color="green">// Get the current speed of the Y axis.</font>
```

```
int mcAxisHome(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisHome(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; <font color="green">// Set the X axis to home.  
</font> mcAxisSetPos(mInst, AxisNumber);
```

```
int mcAxisHomeAll(  
    MINSTANCE mInst);  
  
rc = mc.mcAxisHomeAll(  
    number mInst)  
  
int mInst=0;  
  
<font color="green">// Home all coordinated axes.</font>  
mcAxisHomeAll(mInst);
```

```
int mcAxisHomeComplete(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisHomeComplete(  
    number mInst, number axisId);  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; <font color="green">// Mark the X axis home  
operation as complete.</font> mcAxisHomeComplete(mInst,  
AxisNumber);
```

```
int mcAxisHomeCompleteWithStatus(  
    MINSTANCE mInst, int axisId BOOL success);  
  
rc = mc.mcAxisHomeCompleteWithStatus(  
    number mInst, number axisId number success);  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; <font color="green">// Mark the X axis  
homing operation as finished and that it completed with success.</font>  
mcAxisHomeCompleteWithStatus(mInst, AxisNumber, TRUE);
```

```
int mcAxisIsEnabled(  
    MINSTANCE mInst, int axisId, BOOL *enabled);  
  
enabled, rc = mc.mcAxisIsEnabled(  
    number mInst, number axisId)  
  
<font color="green">// Find all enabled axes.</font> int mInst = 0;  
  
int axisId;  
  
BOOL enabled = FALSE;  
  
for (axisId = 0; axisId < MC_MAX_AXES; axisId++) {  
  
    enabled = false; mcAxisIsEnabled(mInst, axisId, &enabled); if (enabled  
    == TRUE) {  
  
        <font color="green">// Axis is enabled!</font> }  
  
    }  
}
```

```
int mcAxisIsHomed(  
    MINSTANCE mInst, int axisId, BOOL *homed);  
  
homed, rc= mc.mcAxisIsHomed(  
    number mInst, number axisId)  
  
int mInst = 0;  
  
BOOL homed = FALSE; <font color="green">// Get the homed state of Z  
axis.</font> mcAxisIsHomed(mInst, Z_AXIS, &homed); if (rc ==  
MERROR_NOERROR && homed == TRUE) {  
  
    <font color="green">// Z is homed.</font> }
```

```
int mcAxisIsHomin(
    MINSTANCE mInst, int axisId, BOOL *homing);
homing, rc= mc.mcAxisIsHomed(
    number mInst, number axisId)
int mInst = 0;
BOOL homing = FALSE; <font color="green">// Get the homing state of Z
axis.</font> mcAxisIsHoming(mInst, Z_AXIS, &homing); if (rc ==
MERROR_NOERROR && homing == TRUE) {
<font color="green">// Z is in a home operation.</font> }
```

```
int mcAxisIsStill(  
    MINSTANCE mInst, int axisId, BOOL *still);  
  
still, rc = mc.mcAxisIsStill(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; int still = 0;  
  
<font color="green">// Get the is moving state of the axis vluue of 1 is  
stopped.</font> mcAxisIsStill(mInst, AxisNumber, &still);
```

```
int mcAxisMapMotor(  
    MINSTANCE mInst, int axisId, int motorId);  
  
rc = mc.mcAxisMapMotor(  
    number mInst, number axisId, number motorId)  
  
int mInst=0;  
  
int AxisNumber = X_AXIS; int MotorNumber = MOTOR5  
  
<font color="green">// Map motor5 to the X axis.</font>  
mcAxisMapMotor(mInst, AxisNumber, MotorNumber);
```

```
int mcAxisRegister(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisRegister(  
    number mInst, number axisId);  
  
int mInst=0;  
  
for (int a = AXIS0; < AXIS4; a++) {  
    <font color="green">// Register axis0 - axis2 to the core.</font> if(  
    mcAxisRegister(mInst, a) !=  
  
    MERROR_NOERROR No Error.){  
        return(false); }  
    }  
}
```

```
int mcAxisRemoveOverrideAxis(  
    MINSTANCE mInst, int axisId, int axisToOverrideId);  
  
rc = mc.mcAxisRemoveOverrideAxis(  
    number mInst, number axisId, number axisToOverrideId)  
  
int mInst = 0;  
  
int homed = 0;  
  
<font color="green">// Remove axis 8 from the Z axis.</font>  
mcAxisRemoveOverrideAxis( mInst, AXIS_8, ZAXIS );
```

```
int mcAxisRemoveOverrideAxisSync(  
    MINSTANCE mInst, int axisId,  
    int axisToOverrideId);  
  
rc = mc.mcAxisRemoveOverrideAxisSync(  
    number mInst, number axisId, number axisToOverrideId)  
  
int mInst = 0;  
  
int homed = 0;  
  
<font color="green">// Remove axis 8 from the Z axis.</font>  
mcAxisRemoveOverrideAxisSync( mInst, AXIS_8, ZAXIS );
```

```
int mcAxisSetHomeDir(  
    MINSTANCE mInst, int axisId, int dir);  
  
rc = mc.mcAxisSetHomeDir(  
    number mInst, number axisId, number dir)  
  
<font color="green">// Set axis 0 homing directio to POS.</font> int mInst  
= 0;  
  
mcAxisSetHomeDir(mInst, 0, 1);
```

```
int mcAxisSetHomeInPlace(  
    MINSTANCE mInst, int axisId, BOOL homeInPlace);  
  
rc = mc.mcAxisGetHomeInPlace(  
    number mInst, number axisId, number homeInPlace)  
  
<font color="green">// Set the Home In Place flag for the X axis.</font>  
MINSTANCE mInst = 0;  
  
BOOL hip = FALSE;  
  
mcAxisGetHomeInPlace(mInst, X_AXIS, hip);
```

```
int mcAxisSetHomeOffset(  
    MINSTANCE mInst, int axisId, double offset);  
  
rc = mcAxisSetHomeOffset(  
    number mInst, number axisId, number offset);  
  
<font color="green">// Set the home offset for axis 0.</font> int mInst = 0;  
  
mcAxisSetHomeOffset(mInst, 0, 1.5 <font color="green"> /* inches  
 */</font>);
```

```
int mcAxisSetHomeOrder(  
    MINSTANCE mInst, int axisId, int order);  
  
rc = mc.mcAxisSetHomeOrder(  
    number mInst, number axisId, number order)  
  
int mInst=0;  
  
<font color="green">// Set the order of homming so the Z axis will home  
first then the X and Y.</font> mcAxisSetHomeOrder(mInst, Z_AXIS , 0);  
mcAxisSetHomeOrder(mInst, X_AXIS , 1); mcAxisSetHomeOrder(mInst,  
Y_AXIS , 1);
```

```
int mcAxisSetHomeSpeed(  
    MINSTANCE mInst, int axis, double percent);  
  
rc = mc.mcAxisSetHomeSpeed(  
    number mInst, number axis, number percent);  
  
<font color="green">// Set the homing speed for axis 0 as a percentage of  
the max velocity.</font> int mInst = 0;  
  
mcAxisSetHomeSpeed(mInst, 0, 20.5 <font color="green">/* percent  
*/*</font>);
```

```
int mcAxisSetInfoStruct(
```

```
    MINSTANCE mInst, int axisID,
```

```
    axisinfo_t *ainf);
```

N/A

```
struct axisinfo {
```

```
    bool OutOfBandAxis; bool IsStill; // Set high when the axis is not  
    moving int Jogging; // Used to tell to jog...
```

```
    int Homing; // Used to tell the state of the home operation.
```

```
    int Id; // Axis Id bool IsSpindle; // Does this axis control a spindle?
```

```
    bool Enabled; // Is axis enabled?
```

```
    bool SoftlimitEnabled; // Softlimits enabled?
```

```
    double SoftMaxLimit; // Count for the max travel.
```

```
    double SoftMinLimit; // Count for the min travel.
```

```
    bool VelocityMode; // Used to make the axis move at a fixed speed  
    bool BufferJog; double Pos; // Position in user units.
```

```
    double Mpos; // Machine position in user units.
```

```
    int HomeOrder; // The order in which to home the axis.
```

```
};
```

```
int mInst = 0;
```

```
axisinfo_t ainf;
```

```
mcAxisGetInfoStruct(mInst, Y_AXIS, &ainf);<font color="green">// Get
Y_AXIS info structure.</font> ainf.HomeOrder = 1;<font color="green">//
Set Y_AXIS home order to 1.</font> mcAxisSetInfoStruct(mInst,
Y_AXIS, &ainf);<font color="green">// Set Y_AXIS info structure.</font>
```

```
int mcAxisSetMachinePos(  
    MINSTANCE mInst, int axis, double val);  
  
rc = mc.mcAxisSetMachinePos(  
    number mInst, number axis, number val)  
  
<font color="green">// Set axis 0 fisxture offset.</font> int mInst = 0;  
  
mcAxisSetMachinePos(mInst, 0, 0.0 <font color="green"> /* set part zero  
 */</font>);
```

```
int mcAxisSetOverrideAxis(  
    MINSTANCE mInst, int axisTd, int axisToOverrideId);  
  
rc = mc.mcAxisSetOverrideAxis(  
    number mInst, number axisId, number axisToOverrideId)  
  
int mInst = 0;  
  
int homed = 0;  
  
<font color="green">// Set the axis 8 to be the override axis for the Z axis.  
</font> mcAxisSetOverrideAxis(mInst, AXIS_8, ZAXIS);
```

```
int mcAxisSetPos(  
    MINSTANCE mInst, int axisId, double val);  
  
rc = mc.mcAxisSetPos(  
    number mInst, number axisId, number val);  
  
int mInst=0;  
  
double XAxisPos = .5;  
  
int AxisNumber = X_AXIS; <font color="green">// Set the position of the  
X axis by changing the fixture offset.</font> mcAxisSetPos(mInst,  
AxisNumber, XAxisPos);
```

```
int mcAxisSetSoftlimitEnable(  
    MINSTANCE mInst, int axis, int enabel);  
  
rc = mc.mcAxisSetSoftlimitEnable(  
    number mInst, number axis, number enabel)  
  
<font color="green">// Diable softlimits on axis 0</font> MINSTANCE  
mInst = 0;  
  
mcAxisSetSoftlimitEnable(mInst, 0,0);
```

```
int mcAxisSetSoftlimitMax(  
    MINSTANCE mInst, int axisId, double max);  
  
rc = mc.mcAxisSetSoftlimitMax(  
    number mInst, number axisId, number max);  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double max = 20;  
  
mcAxisGetSoftlimitMax( mInst, axis, max);<font color="green">// Set the  
max distance of the softlimit for the Y axis to 20 units.</font>
```

```
int mcAxisSetSoftlimitMin(  
    MINSTANCE mInst, int axisId, double min);  
  
rc = mc.mcAxisSetSoftlimitMin(  
    number mInst, number axisId, number min);  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double min = 20;  
  
mcAxisGetSoftlimitMin( mInst, axis, min);<font color="green">// Set the  
min distance of the softlimit for the Y axis to 20 units.</font>
```

```
int mcAxisSetSpindle(  
    MINSTANCE mInst, int axisId, BOOL spindle);  
  
rc = mc.mcAxisSetSpindle(  
    number mInst, number axisId, number spindle)  
  
<font color="green">// Set axis 6 as a spindle axis.</font> MINSTANCE  
mInst = 0;  
  
mcAxisSetSpindle(mInst, 6, true);
```



Home



Up a level



Previous



Next

mcAxisSetVel

C/C++ Syntax:

```
int mcAxisSetVel(  
    MINSTANCE mInst,  
    int axis,  
    double velocity);
```

Description: Not used at this time.

```
int mcAxisUnmapMotor(  
    MINSTANCE mInst, int axisId, int motor);  
  
rc = mc.mcAxisUnmapMotor(  
    number mInst, number axisId, number motor)  
  
int mInst = 0;  
  
<font color="green">// Remove motor6 from the Y axis.</font>  
mcAxisUnmapMotor(mInst, Y_AXIS, MOTOR6);
```

```
int mcAxisUnmapMotors(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisUnmapMotors(  
    number mInst, number axisId)  
  
int mInst = 0;  
  
<font color="green">// Remove all motors from the Y axis.</font>  
mcAxisUnmapMotors(mInst, Y_AXIS);
```

```
int mcAxisUnregister(  
    MINSTANCE mInst, int axisId);  
  
rc = mc.mcAxisUnregister(  
    number mInst, number axisId)  
  
int mInst=0;  
  
<font color="green">// Remove AXIS7 from frome the system.</font>  
mcAxisUnregister(mInst, AXIS7);
```

```
int mcCtlCheckLicenseFeature(  
    MINSTANCE mInst, const char *licFile, const char *requirement, const  
    char *feature);  
  
rc = mcCtlCheckLicenseFeature(  
    number mInst, string licFile, string requirement, string feature);  
  
<font color="green">// Check the DarwinLic.dat file for feature  
M4_DARWIN</font> MINSTANCE mInst = 0;  
  
int rc = mcCtlCheckLicenseFeature(mInst, "DarwinLic.dat",  
"NF_DARWIN", "M4_DARWIN"); if (rc != MERROR_NOERROR) {  
  
<font color="green">// Feature found!!!</font> return; }
```

```
int mcCtlCleanup(MINSTANCE mInst);
```

N/A

None.

```
<font color="green">// Cleanup core instance 0</font> MINSTANCE  
mInst = 0; int rc = mcCtlCleanup(mInst);
```

```
int mcCtlCloseGCodeFile(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlCloseGCodeFile(  
    number mInst)  
  
int mInst=0;  
  
mcCtlCloseGCodeFile(mInst); <font color="green">// Close the Gcode  
file in controller 0.</font>
```

```
int mcCtlCompileScripts(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlCompileScripts(  
    number mInst);  
  
<font color="green">// Compaile all scripts</font> MINSTANCE mInst =  
0;  
  
int rc = mcCtlCompileScripts(mInst);
```

```
int mcCtlConfigStart(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlConfigStart(  
    number mInst);  
  
<font color="green">// Enter the configure state</font> MINSTANCE  
mInst = 0;  
  
if (mcCtlConfigStart(mInst) == MERROR_NOERROR) {  
  
    <font color="green">// Successfully entered the configure state!</font> }
```

```
int mcCtlConfigStop(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlConfigStop(  
    number mInst);  
  
<font color="green">// Leave the configure state.</font> MINSTANCE  
mInst = 0; int rc = mcCtlConfigStop(mInst); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Successfully exited the configure state.</font> }
```

```
int mcCtlCreateLocalVars(  
    MINSTANCE mInst,  
    const char *lineParams, unsigned long *handle)  
hParams, rc = mc.mcCtlCreateLocalVars(  
    number mInst,  
    string lineParams);  
  
<font color="green">--Example m19 script that takes parameters</font>  
<br/> function m19(hParam) <font color="green">--hParam is a handle to  
all parameters on same line as m19 --The R and P params are optional. M19  
R90 P0 for example.</font> <font color="green">--R is angle from 0 to  
360.</font> <font color="green">--P is direction: 0 == shortest angle, 1 ==  
clockwise, 2 == counterclockwise</font> <font color="green">--Fanuc  
uses S for angle</font> local inst = mc.mcGetInstance() local pcallRet =  
true
```

local stat = 1

local msgPre = "M19 macro says "

local msg = "M19 Completed successfully"

local varR = 0

local varP = 0

if (hParam ~= nil) then local rc, flagR, flagP

```
    flagR, rc = mc.mcCtlGetLocalVarFlag(inst, hParam, mc.SV_R) flagP, rc  
= mc.mcCtlGetLocalVarFlag(inst, hParam, mc.SV_P) if (flagR == 1) then  
<font color="green">--Check that the flag has been set so we do not get an  
unexpected value for mc.SV_R</font> varR = mc.mcCtlGetLocalVar(inst,  
hParam, mc.SV_R) if (varR < 0) or (varR > 360) then <font
```

```
color="green">--It is out of range</font> msg = string.format("varR == " ..  
varR .. " and is out of range (0-360)") mc.mcCtlCycleStop(inst) return
```

```
end
```

```
end
```

```
if (flagP == 1) then <font color="green">--Check that the flag has been  
set so we do not get an unexpected value for mc.SV_P</font> varP =  
mc.mcCtlGetLocalVar(inst, hParam, mc.SV_P) <font color="green">--  
fixup the values to pass to the spindleorient script.</font> if (varP == 0)  
then
```

```
varP = mc.MC_SPINDLE_STOP <font color="green">-- 0</font> elseif  
(varP == 1) then varP = mc.MC_SPINDLE_FWD <font color="green">--  
1</font> elseif (varP == 2) then varP = mc.MC_SPINDLE_REV <font  
color="green">-- -1</font> else
```

```
msg = string.format("varP == " .. varP .. " and is out of range (0-2)")  
mc.mcCtlCycleStop(inst) return
```

```
end
```

```
end
```

```
end
```

```
pcallRet, stat, msg = pcall(spindleorient, varR, varP); <font  
color="green">--Call the spindleorient.mcs script</font>  
mc.mcCtlSetLastError(inst, msgPre .. msg)
```

```
if (pcallRet == false) then mc.mcCtlCycleStop(inst) end
```

end

if (mc.mcInEditor() == 1) then

 --We are testing the script in the editor.
 --Fab up some paramters to pass to our m19()
 function. local inst = mc.mcGetInstance() local hParams, rc = <font
 color="blue">mc.mcCtlCreateLocalVars(inst, "R320 P0")<font
 color="green">--Like we are testing "M19 R320 P0".
 m19(hParams) --Call m19() with hParams. end

```
int mcCtlCycleStart(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlCycleStart(  
    number mInst)  
  
int mInst=0;  
  
<font color="green">// Start controller 0's Gcode file.</font>  
mcCtlCycleStart(mInst);
```

```
int mcCtlCycleStop(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlCycleStop(  
    number mInst)  
  
int mInst=0;  
  
<font color="green">// Stop controller 0's Gcode file.</font>  
mcCtlCycleStop(mInst);
```

```
int mcCtlDryRunToLine(  
    MINSTANCE mInst, int line);  
  
rc = mc.mcCtlDryRunToLine(  
    number mInst, number line)  
  
<font color="green">// Dry run to line 38</font> MINSTANCE mInst = 0;  
  
int rc = mcCtlDryRunToLine(mInst, 38);
```

```
int mcCtlEStop(  
    MINSTANCE mInst);
```

```
rc = mc.mcCtlEStop(  
    number mInst)
```

```
<font color="green">// Put the control in the default state when e-stop is  
asserted.</font> MINSTANCE mInst = 0; int rc = mcCtlEStop(mInst);
```

```
int mcCtlEnable(  
    MINSTANCE mInst, BOOL state);  
  
rc = mc.mcCtlEnable(  
    number mInst, number state)  
  
<font color="green">// Enable the control.</font> MINSTANCE mInst = 0;  
int rc = mcCtlEnable(mInst, TRUE);
```

```
int mcCtlFeedHold(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlFeedHold(  
    number mInst)  
  
int mInst = 0; int rc;  
  
rc = mcCtlFeedHold(mInst); <font color="green">// Pause the motion of  
controller 0.</font> if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Feed hold was successful.</font> }  
}
```

```
int mcCtlFeedHoldState(  
    MINSTANCE mInst, BOOL *InFeedHold);  
  
InFeedHold, rc = mc.mcCtlFeedHoldState(  
    number mInst)  
  
int mInst = 0; int rc;  
  
BOOL fhState; rc = mcCtlFeedHoldState(mInst, &fhState); <font  
color="green">// Pause the motion of controller 0.</font> if (rc ==  
MERROR_NOERROR && fhState == TRUE) {  
  
    <font color="green">// The control is in Feedhold.</font> }  
 
```

```
int mcCtlGcodeExecute(
    MINSTANCE mInst, const char *commands);

rc = mc.mcCtlGcodeExecute(
    number mInst, string commands)

<font color="green">// Execute spindle stop and rapid to 0, 0.</font>
MINSTANCE mInst = 0;

int rc;

rc = mcCtlGcodeExecute(mInst, "M05\nG00 X0 Y0"); if (rc ==
MERROR_NOERROR) {

    <font color="green">// The G code was submitted for processing.</font>
    <font color="green">// However, the function return immediately and does
    not wait on the G code to finish.</font> }
```

```
int mcCtlGcodeExecuteWait(  
    MINSTANCE mInst, const char *commands);  
  
rc = mcCtlGcodeExecuteWait(  
    number mInst, string commands)  
  
<font color="green">// Execute spindle stop and rapid to 0, 0.</font>  
MINSTANCE mInst = 0;  
  
int rc;  
  
rc = mcCtlGcodeExecuteWait(mInst, "M05\nG00 X0 Y0"); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// The G code was submitted for processing and has  
completed.</font> }  
 
```

```
int mcCtlGcodeInterpGetData(
```

```
    MINSTANCE mInst, interperter_t *data);
```

N/A

```
struct Interperter_info {
```

```
    double ModalGroups[MC_MAX_GROUPS]; double FeedRate; double
    SpindleSpeed; int SpindleDirection; BOOL Mist; BOOL Flood; int
    ToolNumber; int HeightRegister; int DiaRegister; };
```

```
typedef struct Interperter_info interperter_t;
```

```
<font color="green">// Get the interpreter information.</font>
MINSTANCE mInst = 0;
```

```
interperter_t data;
```

```
int rc;
```

```
rc = mcCtlGcodeInterpGetData(mInst, &data); if (rc ==
MERROR_NOERROR) {
```

```
<font color="green">// Success!</font> }
```

```
int mcCtlGcodeInterpGetPos(  
    MINSTANCE mInst, int axisId, double *pos);  
  
pos, rc = mc.mcCtlGcodeInterpGetPos(  
    number mInst, number axisId)  
  
<font color="green">// Get the current buffered X axis position.</font>  
MINSTANCE mInst = 0; double pos;  
  
int rc;  
  
rc = int mcCtlGcodeInterpGetPos(mInst, X_AXIS, &pos); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCntrlGetBlockDelete(  
    MINSTANCE mInst, int deleteId, BOOL *val);  
  
val, rc = mc.mcCntrlGetBlockDelete(  
    number mInst, number deleteId)  
  
<font color="green">// Is block delete 0 on?</font> MINSTANCE mInst =  
0;  
  
BOOL val;  
  
int rc = mcCntrlGetBlockDelete(mInst, 0, &val); if (rc ==  
MERROR_NOERROR) {  
  
    if (val == TRUE) {  
  
        <font color="green">// Is block delete 0 is on!</font> } else {  
  
        <font color="green">// Is block delete 0 is off!</font> }  
  
    }  
}
```

```
int mcCtlGetBuild(  
    MINSTANCE mInst, char *buf, size_t bufsize);  
  
build, rc = mc.mcCtlGetBuild(  
    number mInst)  
  
<font color="green">// Get the core's build number</font> MINSTANCE  
mInst = 0; char buildBuf[80];  
  
int rc = mcCtlGetBuild(mInst, buildBuf, sizeof(buildBuf)); if (rc ==  
MERROR_NOERROR) {  
  
    printf("The current build is %s.n", buildBuf); }
```

```
int mcCtlGetComputerID(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCtlGetComputerID(  
    number mInst)  
  
<font color="green">// Get the ID(s) for this host</font> MINSTANCE  
mInst = 0;  
  
int rc = mcCtlGetComputerID(mInst, buf, sizeof(buf)); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Parse buf to get the ID(s)</font> }  
 
```

```
int mcCtlGetCoolantDelay(
    MINSTANCE mInst, double *secs);

sec, rc = mc.mcCtlGetCoolantDelay(
    number mInst)

<font color="green">// Get the coolant delay.</font> MINSTANCE mInst =
0;

double secs;

int rc = mcCtlGetCoolantDelay(mInst, &secs); if (rc ==
MERROR_NOERROR) {

<font color="green">// Success!</font> }
```

```
int mcCtlGetCwd(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCtlGetCwd(  
    number mInst)  
  
<font color="green">// Get current working directory.</font>  
MINSTANCE mInst = 0;  
  
char curDir[255];  
  
int rc = mcCtlGetCwd(mInst, curDir, sizeof(curDir)); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> }  
 
```

```
int mcCtlGetDiaMode(  
    MINSTANCE mInst, BOOL *dia);  
  
dia, rc = mc.mcCtlGetDiaMode(  
    number mInst)  
  
<font color="green">// Get the lathe diameter mode.</font> MINSTANCE  
mInst = 0; BOOL Dia;  
  
int rc = mcCtlGetDiaMode(mInst, &Dia); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetDistToGo(  
    MINSTANCE mInst, int axisId, double *togo);  
  
togo, rc = int mcCtlGetDistToGo(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int axis = Y_AXIS;  
  
double togo=0;  
  
mcCtlGetDistToGo(mInst, axis, &togo);
```

```
int mcCntrlGetEnableFRO(  
    MINSTANCE mInst, BOOL *enable);  
  
eanbled, rc = mc.mcCntrlGetEnableFRO(  
    number mInst)  
  
<font color="green">// Get the current FRO status</font> MINSTANCE  
mInst = 0;  
  
BOOL enabled;  
  
int rc = mcCntrlGetEnableFRO(mInst, &enabled); if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> if (enabled == TRUE) {  
  
        <font color="green">// Feed Rate Override is enabled!</font> } else {  
  
        <font color="green">// Feed Rate Override is disabled!</font> }  
  
    }  
}
```

```
int mcCtlGetFRO(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mc.mcCtlGetFRO(  
    number mInst)  
  
<font color="green">// Get the current FRO</font> MINSTANCE mInst =  
0; double fro;  
  
int rc = mcCtlGetFRO(mInst, &fro); if (rc == MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetGcodeFileName(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCtlGetGcodeFileName(  
    number mInst)  
  
<font color="green">// Get the currently loaded G code file name.</font>  
MINSTANCE mInst = 0;  
  
char gCodeFileName[255];  
  
int rc = mcCtlGetGcodeFileName(mInst, gCodeFileName,  
    sizeof(gCodeFileName)); if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Success.</font> }  
 
```

```
int mcCtlGetGcodeLine(  
    MINSTANCE mInst, int LineNumber, char *buf, long bufSize);  
  
buff, rc = mc.mcCtlGetGcodeLine(  
    number mInst, number LineNumber)  
  
int mInst=0;  
  
int LineNumber = 5; char gline[128];  
  
gline[0] = '0';  
  
mcCtlGetGcodeLine(mInst, LineNumber, gline , 128); <font  
color="green">// Get line number 5 from the Gcode file loaded into  
controller 0.</font>
```

```
int mcCtlGetGcodeLineCount(  
    MINSTANCE mInst, double *count);  
  
count, rc = mc.mcCtlGetGcodeLineCount(  
    number mInst)  
  
int mInst=0;  
  
double count=0; mcCtlGetGcodeLineCount(mInst, &count);
```

```
int mcCtlGetGcodeLineNbr(  
    MINSTANCE mInst, double *val);  
  
val, rc = mc.mcCtlGetGcodeLineNbr(  
    number mInst)  
  
int mInst=0;  
  
double dline=0; mcCtlGetGcodeLineNbr(mInst, &dline); <font  
color="green">//Get the current running line number</font>
```

```
int mcCtlGetInstanceHandle(MINSTANCE mInst, const char *owner,  
HMINSTANCE *hInst);
```

```
hInst, rc = mc.mcCtlGetInstanceHandle(  
    number mInst, string owner) or
```

```
hInst = mc.mcGetInstance(string owner)
```

```
MINSTANCE mInst=0;
```

```
HMINSTANCE hInst;
```

```
mcCtlGetInstanceHandle(mInst, "My Description", &hInst);  
mcCtlCycleStart(hInst); <font color="green">/Will show in the log as  
coming from "My Description"</font>
```

```
int mcCntrlGetLastError(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCntrlGetLastError(  
    MINSTANCE mInst)  
  
int mInst=0;  
  
char error[128]; mcCntrlGetLastError(mInst, error, 128); <font  
color="green">// Receives the last error messge for controller instance 0.  
</font>
```

```
int mcCntrlGetLastLogMsg(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCntrlGetLastLogMsg(  
    number mInst)  
  
int mInst=0;  
  
char msg[128]; mcCntrlGetLastLogMsg(mInst, msg, 128); <font  
color="green">// Receives the last log messge for controller instance 0.  
</font>
```

```
int mcCtlGetLicenseData(  
    MINSTANCE mInst, int index, char *buf, long bufSize);  
  
buf, rc = mc.mcCtlGetLicenseData(  
    number mInst, number index)  
  
<font color="green">// Get the license data at index 1.</font>  
MINSTANCE mInst = 0;  
  
long len;  
  
char *data;  
  
long dataLen = 0;  
  
int index = 1;  
  
int rc = mcCtlGetLicenseDataLen(mInst, index, &dataLen); if (rc ==  
MERROR_NOERROR && dataLen > 0) {  
  
    data = (char *)malloc(dataLen + 1); memset(data, 0, dataLen + 1); rc =  
<font color="blue">mcCtlGetLicenseData(mInst, index, data, dataLen)  
</font>; if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> }  
  
}
```

```
int mcCtlGetLicenseDataLen(  
    MINSTANCE mInst, int index, long *bufSize);  
  
bufSize, rc = mc.mcCtlGetLicenseData(  
    number mInst, number index)  
  
<font color="green">// Get the license data at index 1.</font>  
MINSTANCE mInst = 0;  
  
long len;  
  
char *data;  
  
long dataLen = 0;  
  
int index = 1;  
  
int rc = <font color="blue">mcCtlGetLicenseDataLen(mInst, index,  
&dataLen)</font>; if (rc == MERROR_NOERROR && dataLen > 0) {  
  
    data = (char *)malloc(dataLen + 1); memset(data, 0, dataLen + 1); rc =  
    mcCtlGetLicenseData(mInst, index, data, dataLen); if (rc ==  
    MERROR_NOERROR) {  
  
        <font color="green">// Success!</font> }  
  
    }  
}
```

```
int mcCtlGetLicenseModules(
```

```
    MINSTANCE mInst, unsigned long long *modules)
```

N/A

// Retrieve the licensed modules.

```
MINSTANCE mInst = 0;
```

```
unsigned long long modules = 0; int rc = mcCtlGetLicenseModules(mInst,  
&modules);
```

```
comment, rc = mc.mcCtlGetLocalComment(  
    number mInst,  
    number hVars)
```

```
<font color="green">-- Get local variables.</font> function m700(hVars)
```

```
    local inst = mc.mcGetInstance() <font color="green">-- Get the current  
    instance</font> local nilPoundVar = mc.mcCtlGetPoundVar(inst, 0) local  
    comment = <font color="blue">mc.mcCtlGetLocalComment</font>(inst,  
    hVars) local message = ""
```

```
    if hVars ~= nil then local flag, retval, rc flag, rc =  
        mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag == 1) then  
            retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_A) if rc ==  
                mc.MERROR_NOERROR then message = message .. "A" .. ":" ..  
                tostring(retval) .. ", "
```

```
    end
```

```
    end
```

```
    flag, rc = mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag ==  
        1) then retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_B) if rc ==  
            mc.MERROR_NOERROR then message = message .. "B" .. ":" ..  
            tostring(retval) end
```

```
end
```

```
mc.mcCtlSetLastError(inst, message) end
```

end

if (mc.mcInEditor() == 1) then

--We are testing the script in the editor.
--Fab up some paramters to pass to our m700()
function. local inst = mc.mcGetInstance() local hParams, rc = <font
color="blue">mc.mcCtlCreateLocalVars(inst, "A23 B6 (my
Comment)"--Like we are testing "M700 A23 B6 (my
Comment)". m700(hParams) -- Call m700
with hParams. end

```
retval, rc = mc.mcCtlGetLocalVar(  
    number mInst,  
    number hVars,  
    number varNumber)  
  
<font color="green">-- Get local variables.</font> function m700(hVars)  
  
    local inst = mc.mcGetInstance() -- Get the current instance local  
    nilPoundVar = mc.mcCtlGetPoundVar(inst, hVars, 0) local message = ""  
  
    if hVars ~= nil then local flag, retval, rc flag, rc =  
        mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag == 1) then  
            retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_A) if rc ==  
                mc.MERROR_NOERROR then message = message .. "A" .. ":" ..  
                tostring(retval) .. ", "  
  
        end  
  
    end  
  
    flag, rc = mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag ==  
        1) then retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_B) if rc ==  
            mc.MERROR_NOERROR then message = message .. "B" .. ":" ..  
            tostring(retval) end  
  
    end  
  
    mc.mcCtlSetLastError(inst, message) end
```

end

if (mc.mcInEditor() == 1) then

 m700(nil) -- We can't test this in the editor!
end

```

int mcCtlGetLocalVarFlag(
    MINSTANCE mInst, HMCVARS hVars,
    int varNumber,
    int *retval);

retval, rc = mc.mcCtlGetLocalVarFlag(
    number mInst,
    number hVars,
    number varNumber)

<font color="green">-- Get local variables.</font> function m700(hVars)
    local inst = mc.mcGetInstance() -- Get the current instance local
    nilPoundVar = mc.mcCtlGetPoundVar(inst,0) local message = ""

    if hVars ~= nil then local flag, retval, rc flag, rc =
        mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag == 1) then
        retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_A) if rc ==
        mc.MERROR_NOERROR then message = message .. "A" .. ":" ..
        tostring(retval) .. ", "
    end
    end

    flag, rc = mc.mcCtlGetLocalVarFlag(inst, hVars, mc.SV_A) if (flag ==
    1) then retval, rc = mc.mcCtlGetLocalVar(inst, hVars, mc.SV_B) if rc ==
    mc.MERROR_NOERROR then message = message .. "B" .. ":" ..
    tostring(retval) end
    end

```

```
mc.mcCtlSetLastError(inst, message) end
```

end

if (mc.mcInEditor() == 1) then

 m700(nil) -- We can't test this in the editor!
end

```
int mcCtlGetLogging(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mc.mcCtlGetLogging(  
    number mInst)  
  
<font color="green">// Check is logging is enabled.</font> MINSTANCE  
mInst = 0; BOOL enabled = FALSE; int rc = mcCtlGetLogging(mInst,  
&enabled); if (rc == MERROR_NOERROR && enabled == TRUE) {  
  
    <font color="green">// Logging is enabled!  
  
}  
  
</font>
```

```
int mcCtlGetMachDir(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = int mcCtlGetMachDir(  
    number mInst)  
  
<font color="green">// Get the Mach core installation directory.</font>  
MINSTANCE mInst = 0; char instDir[255];  
  
memset(instDir, 0, sizeof(instDir); int rc = mcCtlGetMachDir(mInst,  
instDir, sizeof(instDir));
```

```
int mcCtlGetMistDelay(  
    MINSTANCE mInst, double *secs);  
  
secs, rc = mcCtlGetMistDelay(  
    number mInst)  
  
<font color="green">// Get the mist delay.</font> MINSTANCE mInst = 0;  
  
double secs;  
  
int rc = mcCtlGetMistDelay(mInst, &secs); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetModalGroup(  
    MINSTANCE mInst, int group, double *val);  
  
val, rc = mc.mcCtlGetModalGroup(  
    number mInst, number group)  
  
<font color="green">// Get the modal code for modal group 1.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcCtlGetModalGroup(mInst, 1); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetMode(  
    MINSTANCE mInst, int *mode);  
  
mode, rc = mcCtlGetMode(  
    number mInst)  
  
<font color="green">// Get the control mode.</font> MINSTANCE mInst =  
0; int mode;  
  
int rc = mcCtlGetMode(mInst, &mode); if (rc == MERROR_NOERROR)  
{  
  
    <font color="green">// Success!</font> }
```

```
int mcCtlGetOffset(  
    MINSTANCE mInst, int axisId, int type,  
    double *offset);  
  
offset, rc = mc.mcCtlGetOffset(  
    number mInst, number axisId, number type)  
  
<font color="green">// </font> MINSTANCE mInst = 0;
```

```
int mcCntrlGetOptionalStop(
    MINSTANCE mInst, BOOL *stop);
stop, rc = mc.mcCntrlGetOptionalStop(
    number mInst)

<font color="green">// See if optional stop is in effect.</font>
MINSTANCE mInst = 0;
BOOL opStop = FALSE;
int rc = mcCntrlGetOptionalStop(mInst, &opStop); if (rc ==
MERROR_NOERROR) {

    if (opStop == TRUE) {

        <font color="green">// Optional Stop is in effect!
    } else {

        <font color="green">// Optional Stop is not in effect!
    }
}

</font></font>
```

```
int mcCntlGetPoundVar(  
    MINSTANCE mInst, int param, double *value);  
  
value, rc = mc.mcCntlGetPoundVar(  
    number mInst, number param)  
  
int mInst=0;  
  
double PoundVar=50;  
  
double Value=0;  
  
<font color="green">;// Get the value of #50.</font>  
mcCntlGetPoundVar(mInst, PoundVar, &Value);
```

```
int mcCtlGetRRO(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mc.mcCtlGetRRO(  
    number mInst)  
  
<font color="green">// Get the current RRO</font> MINSTANCE mInst =  
0; double rro;  
  
int rc = mcCtlGetRRO(mInst, &rro); if (rc == MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetRunTime(  
    MINSTANCE mInst, double *time);  
  
int mcCtlGetRunTime(MINSTANCE mInst, double *time)  
<font color="green">// Get the control run time in seconds.</font>  
MINSTANCE mInst = 0; double time;  
  
int rc = mcCtlGetRunTime(mInst, &time); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCntrlGetSingleBlock(  
    MINSTANCE mInst, BOOL *sbState);  
  
sbState, rc = mc.mcCntrlGetSingleBlock(  
    number mInst)  
  
<font color="green">// Get the state of single block.</font> int mInst=0;  
BOOL IsOn = FALSE; mcCntrlGetSingleBlock(mInst, &IsOn);
```

```
int mcCtlGetState(  
    MINSTANCE mInst, mcState *state);  
  
mcState, rc = mc.mcCtlGetState(  
    MINSTANCE mInst)  
  
<font color="green">// Get the state of controller instance 0.</font>  
MINSTANCE mInst = 0; mcState state;  
  
char stateName[80]; int rc = <font color="blue">mcCtlGetState(mInst,  
&state)</font>; if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> rc =  
    mcCtlGetStateName(mInst, state, stateName, sizeof(stateName)); }
```

```
int mcCtlGetName(
    MINSTANCE mInst, mcState state, char *buf, size_t bufSize);

buf, rc = mcCtlGetName(
    number mInst, number state)

<font color="green">// Get the state name for controller instance 0.</font>
MINSTANCE mInst = 0;

mcState state;

char stateName[80];

int rc = mcCtlGetName(mInst, &state); if (rc == MERROR_NOERROR) {

    <font color="green">// Success!</font> rc = <font
color="blue">mcCtlGetName(mInst, state, stateName,
sizeof(stateName))</font>; }
```

```
int mcCtlGetStats(
```

```
    MINSTANCE mInst, mstats_t *stats);
```

N/A

```
struct mstats {
```

```
    int cannon_buf_depth; int la_cannon_buf_depth; double totalTime;  
    double sessionTime; double spindleTime; long m3count; long m4count;  
    long m6count; double xDistance; double yDistance; double zDistance;  
    double aDistance; double bDistance; double cDistance; };
```

```
typedef struct mstats mstats_t;
```

```
<font color="green">// Get controller statistics.</font> MINSTANCE mInst  
= 0; mstats_t stats;
```

```
int rc = mcCtlGetStats(mInst, &stats); if (rc == MERROR_NOERROR) {
```

```
    printf("M3 count = %dn", stats.m3Count); }
```

```
int mcCtlGetToolOffset(  
    MINSTANCE mInst, int axisId, double *offset);  
  
offset, rc = mc.mcCtlGetToolOffset(  
    number mInst, number axisId)  
  
int mInst=0;  
  
int toolnum = 5; double val = 0; mcCtlGetToolOffset(mInst, Z_AXIS,  
&val); <font color="green">// Get the tool offset distance for the Z axis..  
</font>
```

```
int mcCtlGetUnitsCurrent(  
    MINSTANCE mInst, int *units);  
  
units, rc = mc.mcCtlGetUnitsCurrent(  
    number mInst)  
  
<font color="green">// Get the current machine units.</font>  
MINSTANCE mInst = 0;  
  
int units;  
  
int rc = mcCtlGetUnitsCurrent(mInst, &units); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetUnitsDefault(  
    MINSTANCE mInst, int *units);  
  
units, rc = mc.mcCtlGetUnitsDefault(  
    number mInst)  
  
<font color="green">// Get the default machine units.</font> MINSTANCE  
mInst = 0;  
  
int units;  
  
int rc = mcCtlGetUnitsDefault(mInst, &units); if (rc ==  
MERROR_NOERROR) {  
  
<font color="green">// Success!</font> }
```

```
int mcCtlGetValue(  
    MINSTANCE mInst, int valId, int param, double *value);  
  
value, rc = mc.mcCtlGetValue(  
    number mInst, number valId, number param)  
  
int mInst = 0;  
  
int motor = 1;  
  
double value = 0;  
  
<font color="green">// Get the motor velocity of motor1.</font> int rc =  
mcCtlGetValue(mInst, VAL_MOTOR_VEL, motor, &value);
```

```
int mcCtlGetVersion(  
    MINSTANCE mInst, char *buf, size_t bufSize);  
  
buf, rc = mc.mcCtlGetVersion(  
    number mInst)  
  
<font color="green">// Get the core version string.</font> MINSTANCE  
mInst = 0; char ver[80];  
  
int rc = mcCtlGetVersion(mInst, ver, sizeof(ver)); if (rc ==  
MERROR_NOERROR) {  
  
    printf("The core version is %sn", ver); }
```

```
int mcCtlGotoZero(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlGotoZero(  
    number mInst)  
  
int mInst = 0;  
  
mcCtlGotoZero(mInst); <font color="green">// Move controller instance 0  
to x,y,z,a,b,c zero.</font>
```

```
MINSTANCE mcCtlInit(
```

```
    const char *ProfileName, int ControllerID);
```

N/A

```
MINSTANCE mInst; mInst = mcCtlInit("Mach4Mill", mInst); if (mInst  
>= 0) {
```

```
    <font color="green">// A valid instance has been initialized!</font> }
```

```
int mcCtlIsInCycle(  
    MINSTANCE mInst, BOOL *cycle);  
  
cycle, rc = mc.mcCtlIsInCycle(  
    number mInst)  
  
int mInst = 0;  
  
BOOL InCycle = FALSE; bool RunningFile = false;  
  
int rc = mcCtlIsInCycle(mInst, &InCycle); <font color="green">// See if a  
G code file is running.</font> if(rc == MERROR_NOERROR && InCycle  
== TRUE){  
  
    RunningFile = true; }
```

```
int mcCtlIsStill(
    MINSTANCE mInst, BOOL *still);
still, rc = mcCtlIsStill(
    number mInst)

<font color="green">// See if the control axes are still.</font>
MINSTANCE mInst = 0;
BOOL still;

int rc = mcCtlIsStill(mInst, &still); if (rc == MERROR_NOERROR) {
    if (still == TRUE) {
        <font color="green">// All axes are still.</font> } else {
        <font color="green">// At least one axis is moving!</font> }
    }
}
```

```
int mcCtlLoadGcodeFile(  
    MINSTANCE mInst, const char *FileToLoad);  
  
rc = mc.mcCtlLoadGcodeFile(  
    number mInst, stringFileToLoad)  
  
int mInst=0;  
  
char File[128] = "C:SomeGocdefile.tap"; int rc =  
mcCtlLoadGcodeFile(mInst, &char); <font color="green">// Load  
SomeGocdefile.tap file.</font> if (rc == MERROR_NOERROR) {  
  
    <font color="green">// Success!</font> }
```

```
int mcCtlLoadGcodeString(  
    MINSTANCE mInst, const char *gCode);  
  
rc = mc.mcCtlLoadGcodeString(  
    MINSTANCE mInst, const char *gCode);  
  
<font color="green">// Load G code from a string.</font> MINSTANCE  
mInst = 0;  
  
char *gCode = "%nO1001nG90 G94 G91.1 G40 G49 G17nG20"  
  
int rc = mcCtlLoadGcodeString(mInst, gCode);
```

```
int mcCntlLog(  
    MINSTANCE mInst, const char *message, const char *file, int line);  
  
rc = mc.mcCntlLog(  
    number mInst, string message, string file, number line)  
  
<font color="green">// Load G code from a string.</font> MINSTANCE  
mInst = 0; BOOL test = TRUE;  
  
int rc;  
  
if (test == TRUE) {  
  
    <font color="green">// File and line info in the log.</font>  
    mcCntlLog(mInst, "test == TRUE!", __FILE__, __LINE__); } else {  
  
    <font color="green">// No file and line info in the log.</font>  
    mcCntlLog(mInst, "test == FALSE!", NULL, 0); }
```

```
int mcCtlMachineStateClear(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlMachineStateClear(  
    number mInst)  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); <font color="green">-- push control  
    state to the stack.</font> mc.mcCtlMachineStatePush(inst); <font  
    color="green">-- put machine in G20, and G90 mode.</font>  
    mc.mcCtlGcodeExecuteWait(inst, "G20nG90"); <font color="green">--  
    push control state to the stack.</font> mc.mcCtlMachineStatePush(inst);  
    <font color="green">-- put machine in G21, and G91 mode.</font>  
    mc.mcCtlGcodeExecuteWait(inst, "G21nG91"); <font color="green">--  
    clear the machine state stack and leave the machine in G21 and G91 modes.  
    </font> mc.mcCtlMachineStateClear(inst); end
```

```
int mcCtlMachineStatePop(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlMachineStatePop(  
    number mInst)  
  
<font color="green">-- LUA example</font> function test()  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); <font color="green">-- push control  
    state to the stack saving original modes.</font>  
    mc.mcCtlMachineStatePush(inst); <font color="green">-- put machine in  
    G20, and G90 mode.</font> mc.mcCtlGcodeExecuteWait(inst,  
    "G20nG90"); <font color="green">-- push control state to the stack.</font>  
    mc.mcCtlMachineStatePush(inst); <font color="green">-- put machine in  
    G21, and G91 mode.</font> mc.mcCtlGcodeExecuteWait(inst,  
    "G21nG91"); <font color="green">-- restore the machine state stack to G20  
    and G90 modes.</font> mc.mcCtlMachineStatePop(inst); <font  
    color="green">-- restore the machine state stack to original modes.</font>  
    mc.mcCtlMachineStatePop(inst); end
```

```
int mcCtlMachineStatePush(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlMachineStatePush(  
    number mInst)  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); <font color="green">-- push control  
    state to the stack saving original modes.</font>  
    mc.mcCtlMachineStatePush(inst); <font color="green">-- put machine in  
    G20, and G90 mode.</font> mc.mcCtlGcodeExecuteWait(inst,  
    "G20nG90"); <font color="green">-- push control state to the stack.</font>  
    mc.mcCtlMachineStatePush(inst); <font color="green">-- put machine in  
    G21, and G91 mode.</font> mc.mcCtlGcodeExecuteWait(inst,  
    "G21nG91"); <font color="green">-- restore the machine state stack to G20  
    and G90 modes.</font> mc.mcCtlMachineStatePop(inst); <font  
    color="green">-- restore the machine state stack to original modes.</font>  
    mc.mcCtlMachineStatePop(inst); end
```

```
int mcCtlMacroAlarm(  
    MINSTANCE mInst int error, const char *message);  
  
rc = mc.mcCtlMacroAlarm(  
    number mInst number error string message)  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); mc.mcCtlMacroAlarm(inst, 19, "Test  
    Alram") end
```

```
int mcCtlMacroStop(  
    MINSTANCE mInst int error, const char *message);  
  
rc = mc.mcCtlMacroStop(  
    number mInst number error string message)  
  
<font color="green">-- LUA example</font> function test()  
  
    local inst = mc.mcGetInstance(); mc.mcCtlMacroStop(inst, 19, "Test  
Stop") end
```

```
int mcCtlMdiExecute(  
    MINSTANCE mInst, const char *commands);  
  
rc = mcCtlMdiExecute(  
    number mInst, string commands)  
  
MINSTANCE mInst = 0;  
  
int rc;  
  
<font color="green">// Move the machine back to xy then z zero.</font>  
<font color="green">// Correct example.</font> rc =  
mcCtlMdiExecute(mInst, "G00 G90 X0.0 Y0.0\nZ0.0");  
  
<font color="green">// Incorrect example.</font> rc =  
mcCtlMdiExecute(mInst, "G00 G90 X0.0 Y0.0"); rc =  
mcCtlMdiExecute(mInst, "Z0.0");
```

```
int mcCtlProbeFileClose(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlProbeFileClose(  
    number mInst)  
  
<font color="green">-- probe file close LUA macro example.</font>  
function m112()  
    local inst=mc.mcGetInstance(); local rc =  
    mc.mcCtlProbeFileClose(inst); end  
  
  
  
if (mc.mcInEditor() == 1) then m112() end
```

```
int mcCtlProbeFileOpen(  
    MINSTANCE mInst, const char *fileName, const char *format, BOOL  
    overWrite);  
  
rc = mc.mcCtlProbeFileOpen(  
    number mInst, string fileName, string format, number overWrite);  
  
<font color="green">-- probe file open LUA macro example.</font>  
function m111()  
    local inst = mc.mcGetInstance(); local rc =  
    mc.mcCtlProbeFileOpen(inst, 'probetest.csv', '%.4AXIS_X, %.4AXIS_Y,  
    %.4AXIS_Z\r\n', TRUE); end
```

```
if (mc.mcInEditor() == 1) then m111() end
```

```
int mcCtlProbeGetStrikeStatus(  
    MINSTANCE mInst, BOOL *didStrike);  
  
didstrike, rc = mc.mcCtlProbeGetStrikeStatus(  
    number mInst)  
  
<font color="green">-- probe get strike status example.</font>  
  
local inst = mc.mcGetInstance()  
  
local didStrike, rc = mc.mcCtlProbeGetStrikeStatus(inst)
```

```
int mcCtlReset(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlReset(  
    number mInst)  
  
MINSTANCE mInst = 0; mcCtlReset(mInst); <font color="green">//  
Reset controller instance 0.</font>
```

```
int mcCtlRewindFile(
```

```
    MINSTANCE mInst);
```

```
rc = mc.mcCtlRewindFile(
```

```
    number mInst)
```

```
    MINSTANCE mInst = 0; mcCtlRewindFile(mInst); <font  
color="green">// Rewind controller 0's Gcode file.</font>
```

```
int mcCtlSetBlockDelete(  
    MINSTANCE mInst, int deleteID, BOOL enabled);  
  
rc = mc.mcCtlSetBlockDelete(  
    number mInst, number deleteID, number enabled)  
  
MINSTANCE mInst = 0;  
  
<font color="green">// Enable block delete level 0.</font>  
mcCtlSetBlockDelete(mInst, 0, TRUE);
```

```
int mcCtlSetCoolantDelay(  
    MINSTANCE mInst, double secs);  
  
rc = mc.mcCtlSetCoolantDelay(  
    number mInst, number secs);  
  
<font color="green">// Set 3/4 second coolant delay.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcCtlSetCoolantDelay(mInst, .750);
```

```
int mcCtlSetDiaMode(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mcCtlSetDiaMode(  
    number mInst, number enable);  
  
<font color="green">// Enable diamter mode</font> MINSTANCE mInst =  
0; int rc = mcCtlSetDiaMode(mInst, TRUE);
```

```
int mcCtlSetEnableFRO(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcCtlSetEnableFRO(  
    number mInst, number enable);  
  
<font color="green">// Enable feed rate override</font> MINSTANCE  
mInst = 0;  
  
int rc = mcCtlSetEnableFRO(mInst, TRUE);
```

```
int mcCtlSetFRO(  
    MINSTANCE mInst, double percent);  
  
rc = mc.mcCtlSetFRO(  
    number mInst, number percent);  
  
<font color="green">// Set feed rate override to 150%</font>  
MINSTANCE mInst = 0; int rc = mcCtlSetFRO(mInst, 150.0);
```

```
int mcCtlSetGcodeLineNbr(
```

```
    MINSTANCE mInst, double line);
```

```
int mInst=0; mcCtlSetGcodeLineNbr(mInst, 10); <font color="green">//  
Set the Gcode file to be at line 10.</font>
```

```
int mcCtlSetLastError(  
    MINSTANCE mInst, const char *emsg);  
  
rc = mc.mcCtlSetLastError(  
    number mInst, string emsg)  
  
MINSTANCE mInst = 0;  
  
char *erbuf = "Tool no longer in the spindle"; <font color="green">// Send  
an error message for controller 0.</font> int rc = mcCtlSetLastError(mInst,  
erbuf);
```

```
int mcCtlSetLogging(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcCtlSetLogging(  
    number mInst, number enable)  
  
<font color="green">// Enable logging.</font> MINSTANCE mInst = 0;  
  
int rc = mcCtlSetLogging(mInst, enable);
```

```
int mcCtlSetMistDelay(  
    MINSTANCE mInst, double secs);  
  
rc = mcCtlSetMistDelay(  
    number mInst, number secs);  
  
<font color="green">// Set the mist delay to 3/4 of a second.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcCtlSetMistDelay(mInst, .750);
```

```
int mcCtlSetMode(  
    MINSTANCE mInst, double mode);  
  
rc = mc.mcCtlSetMode(  
    number mInst, number mode)  
  
<font color="green">// Set the interpreter mode to Lathe Diameter.</font>  
MINSTANCE mInst = 0; int rc = mcCtlSetMode(mInst,  
    MC_MODE_LATHE_DIA);
```

```
int mcCtlSetOptionalStop(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mcCtlSetOptionalStop(  
    number mInst number enable)  
  
<font color="green">// Enable optional stop.</font> MINSTANCE mInst =  
0;  
  
int rc = mcCtlSetOptionalStop(mInst, TRUE);
```

mcCtlSetPoundVar(

MINSTANCE mInst, int param, double value);

mcCtlSetPoundVar(

number mInst, number param, number value)

MINSTANCE mInst = 0; int PoundVar = 50;

double Value=21;

// Set the value of #50 to 21. int rc =
mcCtlSetPoundVar(mInst, PoundVar, Value);

```
int mcCtlSetRRO(
```

```
    MINSTANCE mInst, double percent);
```

```
rc = mc.mcCtlSetRRO(
```

```
    number mInst, number percent);
```

```
<font color="green">// Set feed rapid override to 50%</font>
MINSTANCE mInst = 0; int rc = mcCtlSetRRO(mInst, 50.0);
```

```
int mcCtlSetResetCodes(  
    MINSTANCE mInst, const char *resetCodes);  
  
rc = mc.mcCtlSetResetCodes(  
    number mInst, string resetCodes)  
  
<font color="green">// Set the G code used to reset the controller instance  
0.</font> MINSTANCE mInst = 0;  
  
int rc = mcCtlSetResetCodes(mInst, "G40 G20 G90 G52 X0 Y0  
Z0\nG92.1 G69");
```

```
int mcCtlSetSingleBlock(  
    MINSTANCE mInst, BOOL enable);  
  
<font color="green">// Enable single block mode.</font> MINSTANCE  
mInst = 0;  
  
mcCtlSetSingleBlock(mInst, TRUE);
```

```
int mcCtlSetStats(  
    MINSTANCE mInst, mstats_t *stats);
```

N/A

```
<font color="green">// </font> MINSTANCE mInst = 0;
```

```
int mcCtlSetValue(  
    MINSTANCE mInst, int valId, int param, double value);  
  
rc = mcCtlSetValue(  
    number mInst, number valId, number param, number value)  
    MINSTANCE mInst = 0; int motor = 1;  
  
value=1000.0;  
  
<font color="green">// Set the Velocity of the motor to 1000.0 .</font> in  
rc = mcCtlSetValue(mInst, VAL_MOTOR_VEL, motor, value);
```

```
int mcCtlStartMotionDev(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlStartMotionDev(  
    number mInst)  
  
<font color="green">// Start the selected motion device.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcCtlStartMotionDev(mInst);
```

```
int mcCtlStopMotionDev(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlStopMotionDev(  
    number mInst);  
  
<font color="green">// Stop the selected motion device.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcCtlStopMotionDev(mInst);
```

```
int mcCtlToolChangeManual(  
    MINSTANCE mInst);  
  
rc = mc.mcCtlToolChangeManual(  
    number mInst)
```

```
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance(); local rc = mc.mcCtlToolChangeManual(inst);
```

```
int mcCtlWaitOnCycleStart(  
    MINSTANCE mInst, const char *msg, int timeOutMs);  
  
rc = mc.mcCtlWaitOnCycleStart(  
    number mInst, string msg, number timeOutMs);  
  
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance();  
  
local rc = mc.mcCtlWaitOnCycleStart(inst, "Please press Cycle Start.",  
1000);
```

```
int mcDeviceGetHandle(  
    MINSTANCE mInst, int devid, HMCDEV *hDev);  
  
hDev, rc = mc.mcDeviceGetHandle(  
    number mInst, number devid)  
  
int mInst = 0;  
  
int devid = 0;  
  
HMCDEV hDev;  
  
//See if we can find a device with an I of zero int rc = <font  
color="blue">mcDeviceGetHandle(mInst, devid, &hDev);</font> if (rc ==  
MERROR_NOERROR) {  
  
    <font color="green">// We found it!</font> }  
}
```

```
int mcDeviceGetInfo(
    HMCDEV hDev, char *nameBuf, size_t nameBuflen, char *descBuf,
    size_t descBuflen, int *type, int *id);

nameBuf, descBuf, type, id, rc = mc.mcDeviceGetInfo(HMCDEV hDev}

int mInst=0;

HMCSIG hDev = 0;

devinfo_t devinf;

char nameBuf[80];

char descBuf[80];

int type = 0;

int id = 0;

//Look for all the IO devices and get their devinfo_t struct while
(mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev) ==
MERROR_NOERROR) {

    if (hSig != 0) {

        <font color="blue">mcDeviceGetInfo(hDev, nameBuf, sizeof(nameBuf),
descBuf, sizeof(descBuf), &type, &id);</font> <font color="green">// do
something with the device info.</font> } else {

        break; }

}
```

```
int mcDeviceGetInfoStruct(  
    HMCSIG hDev, devinfo_t *devinf);  
  
N/A  
  
struct devinfo {  
    char devName[80]; char devDesc[80]; int devType; int devId; };  
  
typedef struct devinfo devinfo_t;  
  
int mInst=0;  
  
HMCSIG hDev = 0; devinfo_t devinf;  
  
//Look for all the IO devices and get their devinfo_t struct while  
(mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev, &hDev) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcDeviceGetInfoStruct(hDev, devinf); </font> <font  
        color="green">// Do something with the device info.</font> } else {  
  
        break; }  
  
    }  
}
```

```
int mcDeviceGetNextHandle(
    MINSTANCE mInst, int devtype, HMCDEV startDev, HMCDEV
*hDev);

hDev, rc = mc.mcDeviceGetNextHandle(
    number mInst, number devtype, number startDev)

int mInst=0;

HMCDEV hDev;

<font color="green">/>Look for all the IO devices</font> while (<font
color="blue">mcDeviceGetNextHandle(mInst, DEV_TYPE_IO, hDev,
&hDev)</font> == MERROR_NOERROR) {

    if (hSig != 0) {

        <font color="green">/> Do something with hDev.</font> } else {

            break; }

}
```

mcDeviceRegister(

 MINSTANCE mInst, HMCPLUG plugin, const char *DeviceName,
 const char *DeviceDesc, int Type, HMCDEV *hDev);

N/A

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {

 m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime
 = .001;

 mcDeviceRegister(m_cid, m_id, "Sim0",
 "Simulation Device", DEV_TYPE_MOTION | DEV_TYPE_IO,
 &m_hDev);

 mcRegisterIo(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,
 &m_Input0); mcRegisterIo(m_hDev, "Output0", "Output0",
 IO_TYPE_OUTPUT, &m_Output0);

 if (!m_timer->Start(10, wxTIMER_ONE_SHOT)) {

 wxMessageBox(wxT("Timer could not start!"), wxT("Timer")); }

}

```
int mcFileHoldAquire(  
    MINSTANCE mInst, const char *reason, int JogAxisBits);  
  
rc = mc.mcFileHoldAquire(  
    number mInst, string reason, number JogAxisBits)  
  
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance();  
  
local rc = mc.mcFileHoldAquire(inst, "My hold reason", 0);
```

```
int mcFileHoldReason(
```

```
    MINSTANCE mInst, char *buf, long bufSize);
```

```
reason, rc = mc.mcFileHoldReason(
```

```
    number mInst)
```

```
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance(); local rc = mc.mcFileHoldAquire(inst, "My hold  
reason", 0); loacl reason
```

```
reason, rc = mc.mcFileHoldReason(inst);
```

```
int mcFileHoldRelease(  
    MINSTANCE mInst);  
  
rc = mc.mcFileHoldRelease(  
    number mInst)  
  
<font color="green">-- LUA example</font> local inst =  
mc.mcGetInstance(); local rc = mc.mcFileHoldAquire(inst, "My hold  
reason", 0); loacl reason  
  
reason, rc = mc.mcFileHoldReason(inst); <font color="green">-- Do some  
script magic...</font> rc = mc.mcFileHoldRelease(inst); <font  
color="green">-- G code processing resumes.</font>
```

```
int mcFixtureLoadFile(  
    MINSTANCE mInst, const char *FileToLoad);  
  
rc = mc.mcFixtureLoadFile(  
    number mInst, string FileToLoad);  
  
<font color="green">// Load a fixture table file.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcFixtureLoadFile(mInst, "MyFixtureTable.dat");
```

```
int mcFixtureSaveFile(  
    MINSTANCE mInst);  
  
rc = mc.mcFixtureSaveFile(  
    number mInst),  
  
<font color="green">// Save the fixture table file.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcFixtureSaveFile(mInst);
```

```
int mcGuiGetWindowHandle(  
    MINSTANCE mInst, void **handle);
```

N/A

```
<font color="green">// </font> MINSTANCE mInst = 0; void *guiHandle;  
rc = mcGuiGetWindowHandle(mInst, &guiHandle);
```

```
int mcGuiSetCallback(  
    MINSTANCE mInst, void *fp);
```

N/A

```
<font color="green"> /* In the GUI code */ </font> MCP_API <font  
color="blue">int</font> MCP_APIENTRY mcGUIMsg(MINSTANCE  
mInst, <font color="blue">long</font> msg, <font  
color="blue">long</font> wparam, <font color="blue">long</font>  
lparam) {  
  
    <font color="green"> // Process messages...</font>  
    return(MERROR_NOERROR); }
```

```
MINSTANCE mInst = 0; mcGuiSetCallback(mInst, &mcGUIMsg);
```

```
int mcGuiSetFocus(  
    MINSTANCE mInst, BOOL focus);
```

N/A

// Set focus to the GUI's main window
MINSTANCE mInst = 0; int rc = mcGuiSetFocus(mInst, TRUE);

```
int mcGuiSetWindowHandle(  
    MINSTANCE mInst, void *handle);
```

N/A

```
<font color="green">// Set the GUI main window handle.</font>  
MINSTANCE mInst = 0; mcGuiSetWindowHandle(0, this);
```

```
mcIoGetHandle(  
    MINSTANCE mInst, const char *path, HMCIO *hIo);  
  
hIo, rc = mc.mcIoGetHandle(  
    number mInst, string path)  
  
HMCIO hIo;  
  
char *path = "Sim0/Input0";  
  
<font color="blue">mcIoGetHandle(m_cid, path, &hIo);</font>
```

```

int mcIoGetInfoStruct(
    HMCIO hIo, ioinfo_t *ioinf);

ioinf, rc = mc.mcIoGetInfoStruct(
    number hIo)

struct ioinfo {
    char ioName[80]; char ioDesc[80]; int ioType; HMCDEV ioDev;
    HMCSIG ioMappedSignals[MAX_MAPPED_SIGNAL]; void
    *ioUserData; int ioInput; };

typedef struct ioinfo ioinfo_t;

HMCDEV hDev = 0;

devinfo_t devinf;

int rc;

<font color="green">// Get all IO information from every registered device.
</font> while (mcDeviceGetNextHandle(0, DEV_TYPE_IO, hDev,
&hDev) == MERROR_NOERROR) {

    if (mcDeviceGetInfoStruct(hDev, &devinf) == MERROR_NOERROR) {

        HMCIO hIo = 0; while (mcIoGetNextHandle(hDev, hIo, &hIo) ==
MERROR_NOERROR) {

            ioinfo_t ioinf; rc = <font color="blue">mcIoGetInfoStruct(hIo, &ioinf);
</font> if (rc ==

```

```
MERROR_NOERROR No Error.) {  
    <font color="green">// IO information successfully retrieved.</font> }  
}  
}  
}
```

```

mcIoGetNextHandle(
    HMCDEV hDev, HMCIO startIo HMCIO *hIo)
hIo, rc = mc.mcIoGetNextHandle(
    number hDev, number startIo)
HMCDEV hDev = 0;
devinfo_t devinf;
int rc;

<font color="green">// Get all IO information from every registered device.
</font> while (mcDeviceGetNextHandle(0, DEV_TYPE_IO, hDev,
&hDev) == MERROR_NOERROR) {

    if (mcDeviceGetInfoStruct(hDev, &devinf) == MERROR_NOERROR) {

        HMCIO hIo = 0; while (<font color="blue">mcIoGetNextHandle(hDev,
hIo, &hIo)</font> == MERROR_NOERROR) {

            ioinfo_t ioinf; rc = mcIoGetInfoStruct(hIo, &ioinf); if (rc ==
MERROR_NOERROR) {

                <font color="green">// IO information successfully retrieved.</font> }

            }

        }

    }

}

```

```
mcIoGetState(  
    HMCIO hIo, BOOL *state);  
  
state, rc = mcIoGetState(  
    number hIo)  
  
simControl::simControl(MINSTANCE mInst, HMCPLUG id) {  
  
    m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime  
    = .001; bool oldstate = false; bool newstate = false;  
  
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",  
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);  
  
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,  
    &m_Input0); <font color="blue">mcIoGetState(m_Input0, &oldstate);  
    </font> newstate = simGetIO(0); if (newstate != oldstate) {  
  
        mcIoSetState(m_Input0, newstate); }  
    }  
}
```

```
mcIoGetType(  
    HMCIO hIo, int *type);  
  
type, rc = mcIoGetType(  
    number hIo)  
  
simControl::simControl(MINSTANCE mInst, HMCPLUG id) {  
  
    m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime  
    = .001; bool oldstate = false; bool newstate = false;  
  
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",  
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);  
  
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,  
    &m_Input0); <font color="blue">mcIoGetType(m_Input0, &type);</font>  
    <font color="green">// type will equal IO_TYPE_INPUT</font> }
```

```
mcIoGetUserData(  
    HMCIO hIo, void **data);
```

N/A

```
struct myData {  
    int myPortNumber; int myPinNumber; };
```

```
void GetUserData(void) {  
  
    MINSTANCE mInst = 0; HMCIO hIo; void *data; if  
(mcIoGetHandle(mInst, "Sim0/Input0", &hIo) == MERROR_NOERROR)  
{  
  
    <font color="blue">mcIoGetUserData(hIo, &data);</font> <font  
color="green">// type cast the pointer</font> struct myData *mData =  
(struct myData *)data; }  
  
}
```

```
mcIoIsEnabled(  
    HMCIO hIo, BOOL *enabled);  
  
enabled, rc = mcIoIsEnabled(  
    number hIo)  
  
HMCIO hIo;  
  
char *path = "Sim0/Input0"; BOOL enabled; mcIoGetHandle(m_cid, path,  
&hIo);<font color="green">//Get the handle</font> mcIoIsEnabled(hIo ,  
&enabled);<font color="green">//Get the enabled state of the IO</font>
```

mcIoRegister(

 HMCDEV hDev, const char *IoName, const char *IoDesc, int Type,
 HMCIO *hIo);

N/A

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {

 m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime
 = .001;

 mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",
 DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);

 mcIoRegister(m_hDev, "Input0", "Input0",
 IO_TYPE_INPUT, &m_Input0); mcIoRegister(m_hDev, "Output0",
 "Output0", IO_TYPE_OUTPUT, &m_Output0);

 if (!m_timer->Start(10, wxTIMER_ONE_SHOT)) {

 wxMessageBox(wxT("Timer could not start!"), wxT("Timer")); }

}

```
mcIoSetDesc(  
    HMCIO hIo, const char *desc);  
  
rc = mc.mcIoSetDesc(  
    number hIo, string desc)  
  
HMCIO hIo;  
  
char *path = "Sim0/Input0";  
  
if (mcIoGetHandle(m_cid, path, &hIo) == MERROR_NOERROR) {  
    <font color="blue">mcIoSetDesc(hIo, "my mew description);</font> }  
}
```

```
mcIoSetName(  
    HMCIO hIo, const char *name);  
  
rc = mc.mcIoSetName(  
    number hIo, string name)  
  
HMCIO hIo;  
  
char *path = "Sim0/Input0";  
  
if (mcIoGetHandle(m_cid, path, &hIo) == MERROR_NOERROR) {  
  
    <font color="blue">mcIoSetName(hIo, "Limit0++");</font> <font  
    color="green">// The name is changed from Input0 to Limit0++</font> }  
}
```

```
mcIoSetState(  
    HMCIO hIo,  
    bool state);  
  
simControl::simControl(MINSTANCE mInst, HMCPLUG id) {  
  
    m_cid = mInst; m_id = id;  
  
    m_timer = new simTimer(this); m_cycletime = .001; bool oldstate =  
    false; bool newstate = false;  
  
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",  
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);  
  
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,  
    &m_Input0); <font color="blue">mcIoGetState(m_Input0, &oldstate);  
    </font> newstate = simGetIO(0); if (newstate != oldstate) {  
  
        mcIoSetState(m_Input0, newstate); }  
    }  
}
```

```
mcIoSetType(  
    HMCIO hIo,  
    int type);  
  
rc = mcIoSetType(  
    number hIo  
    number type)  
  
simControl::simControl(MINSTANCE mInst, HMCPLUG id) {  
    m_cid = mInst; m_id = id;  
  
    m_timer = new simTimer(this); m_cycletime = .001; bool oldstate =  
    false; bool newstate = false;  
  
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",  
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);  
  
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,  
    &m_Input0); <font color="blue">mcIoSetType(m_Input0,  
    IO_TYPE_OUTPUT);</font> <font color="green">// change the type to  
    IO_TYPE_OUTPUT</font> }  
}
```

```
mcIoSetUserData(  
    HMCIO hIo, void **data);
```

N/A

```
struct myData {  
    int myPortNumber; int myPinNumber; };
```

```
struct myData data;
```

```
void GetUserData(void) {  
  
    MINSTANCE mInst = 0; HMCIO hIo; struct myData data;  
    data.myPortNumber = 1; data.myPinNumber = 12; if  
(mcIoGetHandle(mInst, "Sim0/Input0", &hIo) == MERROR_NOERROR)  
{  
  
    <font color="blue">mcIoSetUserData(hIo, &data);</font> }  
  
}
```

mcIoSyncSignal(

 HMCIO hIo,

 BOOL state);

N/A

N/A

```
mcIoUnregister(  
    HMCDEV hDev, HMCIO hIo);
```

N/A

N/A

```
int mcJogAbsStart(  
    MINSTANCE mInst, int axisId, double pos);  
  
rc = mc.mcJogAbsStart(  
    number mInst, number axisId, number pos)  
  
MINSTANCE mInst = 0;  
  
int axis = Z_AXIS;  
  
double JogToPos = 5.0; <font color="green">// Jog controller 0 to position  
5.0 in the Z axis.</font> mcJogAbsStart(mInst, axis, Jogpos);
```

```
int mcJogAbsStop(  
    MINSTANCE mInst, int axisId, double incr);  
  
rc = mc.mcJogAbsStop(  
    number mInst, number axisId, number incr)  
  
<font color="green">// Stop the incremental jog on the Z axis.</font>  
MINSTANCE mInst = 0; int axis = Z_AXIS;  
  
double Joginc = .100; int rc = mcJogAbsStop(mInst, axis, Joginc);
```

```
int mcJogGetAccel(  
    MINSTANCE mInst, int axisId, double *percent);  
  
percent, rc = mc.mcJogGetAccel(  
    number mInst, number axis)  
  
<font color="green">// Get the jog accel percentage for the X axis.</font>  
MINSTANCE mInst = 0;  
  
double accel;  
  
int rc = mcJogGetAccel(mInst, X_AXIS, &accel);
```

```
int mcJogGetFeeRate(  
    MINSTANCE mInst, int axisId, double *feedRate);  
  
feedRate, rc = mc.mcJogSetFeedRate(  
    number mInst, number axisId)  
  
<font color="green">// Set the jog rate to 75% of the Z axis maximum  
velocity.</font> MINSTANCE mInst = 0;  
  
int axis = Z_AXIS;  
  
double feedRate = 0.0;  
  
int rc = mcJogSetUnitsMode(mInst, axis, MC_UNITS_INCH); rc =  
mcJogSetFeedRate(mInst, axis, &feedRate); <font color="green">// Get the  
feed rate in inches per minute.</font>
```

```
mcJogGetFollowMode(  
    MINSTANCE mInst, double *mode_on);  
  
int mInst=0;  
  
double jfstat=0; mcJogGetFollowMode(mInst, &jfstate);
```

```
int mcJogGetInc(  
    MINSTANCE mInst, int axisId, double *increment);  
  
increment, rc = mc.mcJogGetInc(  
    number mInst, number axisId)  
  
<font color="green">// Get the jog increment for the X axis.</font>  
MINSTANCE mInst = 0;  
  
double inc;  
  
int rc = mcJogGetIncAccel(mInst, X_AXIS, &inc);
```

```
int mcJogGetRate(  
    MINSTANCE mInst, int axisId, double *percent);  
  
percent, rc = mc.mcJogGetRate(  
    number mInst, number axisId)  
  
<font color="green">// Get the current jog rate for the X axis</font>  
MINSTANCE mInst = 0;  
  
double jogRate;  
  
int rc = mcJogGetRate(mInst, X_AXIS, &jogRate);
```

```
int mcJogGetTraceEnable(MINSTANCE mInst, BOOL *enable);  
enable, rc = mc.mcJogGetTraceEnable(number mInst)  
<font color="green">// Get jog trace enable status.</font> MINSTANCE  
mInst = 0;  
  
int enabled = MC_FALSE; int rc = mcJogGetTraceEnable(mInst,  
&enabled);
```

```
int mcJogGetType(MINSTANCE mInst, int axisId, int *type);

type, rc = mc.mcJogGetType(
    number mInst, number axisId)

<font color="green">// Get the X axis jog type. </font> MINSTANCE
mInst = 0;

int type = 0;

int rc = mcJogGetType(mInst, X_AXIS, &type);
```

```
mcJogGetUnitsMode(
```

```
    MINSTANCE mInst, int axisId, int *mode);
```

```
int mInst = 0; int mode = 0; mcJogGetUnitsMode(mInst, X_AXIS,  
    &mode);
```

```
int mcJogGetVelocity(  
    MINSTANCE mInst, int axisId, double *vel);  
  
vel, rc = mc.mcJogGetVelocity(  
    number mInst, number axisId),  
  
<font color="green">// Get the current jog velocity setting for the X  
axis</font> MINSTANCE mInst = 0; double jogVel;  
  
int rc = mcJogGetVelocity(mInst, X_AXIS, &jogVel);
```

```
int mcJogIncStart(  
    MINSTANCE mInst, int axisId, double dist);  
  
rc = mc.mcJogIncStart(  
    number mInst, number axisId, number dist)  
  
MINSTANCE mInst = 0; int axis = Z_AXIS;  
  
double Joginc = .100; <font color="green">// Jog controller 0 .1 in the Z  
axis.</font> mcJogIncStart(mInst, axis, Joginc);
```

```
int mcJogIncStop(  
    MINSTANCE mInst, int axisId, double incr);  
  
rc = mc.mcJogIncStop(  
    number mInst, number axisId, number incr)  
  
<font color="green">// Stop the incremental jog on the Z axis.</font>  
MINSTANCE mInst = 0; int axis = Z_AXIS;  
  
double Joginc = .100; int rc = mcJogIncStop(mInst, axis, Joginc);
```

```
int mcJogIsJogging(  
    MINSTANCE mInst, int axisId, BOOL *jogging);  
  
jogging, rc = mc.mcJogIsJogging(  
    number mInst, number axisId)  
  
<font color="green">// See if the X axis is jogging.</font> MINSTANCE  
mInst = 0;  
  
BOOL jogging = FALSE;  
  
int rc = mcJogIsJogging(mInst, X_AXIS, &jogging);
```

```
int mcJogIsStopping(
    MINSTANCE mInst, int axisId, BOOL *stopping);
    stopping, rc = mc.mcJogIsStopping(
        number mInst, number axisId)

<font color="green">// See if the X axis is stopping.</font> MINSTANCE
mInst = 0;

BOOL stopping = FALSE;
int rc = mcJogIsStopping(mInst, X_AXIS, &stopping);
```

```
int mcJogSetAccel(  
    MINSTANCE mInst, int axisId, double percent);  
  
rc = mc.mcJogSetAccel(  
    number mInst, number axisId, double percent)  
  
<font color="green">// Set the X axis jog accel percentage to 75%.</font>  
MINSTANCE mInst = 0;  
  
int mcJogSetAccel(mInst, X_AXIS, 75);
```

```
int mcJogSetFeeRate(  
    MINSTANCE mInst, int axisId, double feedRate);  
  
rc = mc.mcJogSetFeedRate(  
    number mInst, number axisId, number feedRate)  
  
<font color="green">// Set the jog rate to 75% of the Z axis maximum  
velocity.</font> MINSTANCE mInst = 0;  
  
int axis = Z_AXIS;  
  
int rc = mcJogSetUnitsMode(mInst, axis, MC_UNITS_INCH); rc =  
mcJogSetFeedRate(mInst, axis, 20); <font color="green">// 20 inches per  
minute.</font>
```

```
mcJogSetFollowMode(  
    MINSTANCE mInst, double mode_on);  
  
int mInst=0; mcJogGetFollowMode(mInst, MC_ON);
```

```
int mcJogSetInc(  
    MINSTANCE mInst, int axisId, double increment);  
  
rc = mc.mcJogSetInc(  
    number mInst, number axisId, number increment)  
  
<font color="green">// Set the X axis jog increment to .010</font>  
MINSTANCE mInst = 0;  
  
int rc = mcJogSetInc(mInst, X_AXIS, .010);
```

```
int mcJogSetRate(  
    MINSTANCE mInst, int axisId, double percent);  
  
rc = mc.mcJogSetRate(  
    number mInst, number axisId, number percent)  
  
<font color="green">// Set the jog rate to 75% of the Z axis maximum  
velocity.</font> MINSTANCE mInst = 0; int axis = Z_AXIS;  
  
int rc = mcJogSetRate(mInst, axis, 75);
```

```
int mcJogSetTraceEnable(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcJogSetTraceEnable(  
    number mInst, number enable)  
  
<font color="green">//Set jog tracing.</font> MINSTANCE mInst = 0;  
  
int rc = mcJogSetTraceEnable(mInst, MC_TRUE);
```

```
int mcJogSetType(MINSTANCE mInst, int axisId, int type);  
rc = mc.mcJogSetType(  
    number mInst, number axisId, number type)  
<font color="green">// Set the X axis jog type to velocity mode. </font>  
MINSTANCE mInst = 0;  
int rc = mcJogSetType(mInst, X_AXIS, MC_JOG_TYPE_VEL);
```

```
mcJogSetUnitsMode(  
    MINSTANCE mInst, int axisId, int mode);  
  
int mInst = 0;  
  
mcJogSetUnitsMode(mInst, X_AXIS, MC_UNITS_INCH);
```

```
int mcJogVelocityStart(  
    MINSTANCE mInst, int axisId, double dir);  
  
rc = mc.mcJogVelocityStart(  
    number mInst, number axisId, number dir);  
  
int mInst=0;  
  
int axis = Z_AXIS;  
  
int rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS); <font  
color="green">// Start Z axis jogging for 5 sec.</font> if (rc ==  
MERROR_NOERROR) {  
  
    Sleep(5000); }  
  
rc = mcJogVelocityStart(mInst, axis, MC_JOG_POS); <font  
color="green">// Reverse axis and jog the axis Back.</font> if (rc ==  
MERROR_NOERROR) {  
  
    Sleep(5000); }  
  
mcJogVelocityStop(mInst, axis); <font color="green">// Stop the axis.  
</font>
```

```
mcJogVelocityStop(
```

```
    MINSTANCE mInst, int axisId);
```

```
mcJogVelocityStop(
```

```
    number mInst, number axisId)
```

```
MINSTANCE mInst = 0; int axis = Z_AXIS;
```

```
int rc;
```

```
<font color="green">// Jog axis at 10 % max velocity.</font> rc =  
mcJogSetRate(mInst, axis, 10); rc = mcJogVelocityStart(mInst, axis,  
MC_JOG_POS); if (rc == MERROR_NOERROR) {
```

```
    Sleep(5000); }
```

```
<font color="green">// Stop the axis.</font> rc =  
mcJogVelocityStop(mInst, axis);
```

```
int mcMotionClearPlanner(  
    MINSTANCE mInst);  
  
rc = mc.mcMotionClearPlanner(  
    number mInst)  
  
<font color="green">// Clear the planner.</font> MINSTANCE mInst = 0;  
  
int rc = mcMotionClearPlanner(mInst);
```

mcMotionCyclePlanner

C/C++ Syntax:

```
int mcMotionCyclePlanner(MINSTANCE mInst);
```

LUA Syntax:

N/A

Description: Cycle the core planner.

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

This function is deprecated in favor of mcMotionCyclePlannerEx().

Usage:

None .

```
int mcMotionCyclePlannerEx(  
    MINSTANCE mInst, execution_t *exInfo);
```

N/A

// Cycle the core planner. MINSTANCE mInst = 0; execution_t exInfo; int rc = mcMotionCyclePlannerEx(mInst, &exInfo); // For more information, see the Sim plugin example.


```
int mcMotionGetAbsPos(
```

```
    MINSTANCE mInst, int motorId, double *val);
```

N/A

None.

```
int mcMotionGetAbsPosFract(  
    MINSTANCE mInst, int motorId, double *val);
```

N/A

None.

```
int mcMotionGetBacklashAbs(  
    MINSTANCE mInst, int motorId, double *pos);
```

N/A

None.

```
int mcMotionGetBacklashInc(  
    MINSTANCE mInst, int motorId, double *pos);
```

N/A

None.

```
int mcMotionGetIncPos(
```

```
    MINSTANCE mInst, int motorId, double *val);
```

N/A

None.

```
int mcMotionGetMoveID(
```

```
    MINSTANCE mInst, int motorId, long *val);
```

N/A

None.

```
int mcMotionGetPos(  
    MINSTANCE mInst, int motorId, double *pos);  
  
pos, rc = mc.mcMotionGetPos(  
    number mInst, number motorId)  
  
<font color="green">// Get the motor 0 position.</font> MINSTANCE  
mInst = 0;  
  
double pos;  
  
int rc = mcMotionGetPos(mInst, 0, &pos);
```

```
int mcMotionGetProbeParams(  
    MINSTANCE mInst, probe_t *probeInfo);
```

N/A

```
<font color="green">// Get the probing parameters from the core.</font>  
MINSTANCE mInst = 0; probe_t pInfo; int  
mcMotionGetProbeParams(mInst, &pInfo);
```

```
int mcMotionGetRigidTapParams(  
    MINSTANCE mInst, tap_t *tapInfo);
```

N/A

```
<font color="green">// Get the tpping parameters from the core.</font>  
MINSTANCE mInst = 0; tap_t tapInfo; int rc =  
mcMotionGetRigidTapParams(mInst, &tapInfo);
```

```
int mcMotionGetSyncOutput(
```

```
    MINSTANCE mInst, int outputQueue, HMCIO *hIo, BOOL *state);
```

N/A

// Retrieve the coordinated outputs/

```
MINSTANCE mInst = 0; execution_t ex;
```

```
mcMotionCyclePlannerEx(mInst, &ex); if (ex.exOutputQueue !=  
EX_NONE) {
```

```
    HMCIO hIo; BOOL state; while (mcMotionGetSyncOutput(m_cid,  
ex.exOutputQueue, &hIo, &state) == MERROR_NOERROR) {
```

// Pair setting the output along with the moves in
(execution_t)ex. }

```
}
```

```
int mcMotionGetThreadParams(  
    MINSTANCE mInst, thread_t *threadInfo);  
  
<font color="green">// </font> MINSTANCE mInst = 0;  
  
thread_t threadInfo;  
  
int mcMotionGetThreadParams(mInst, &threadInfo);
```

```
int mcMotionGetThreadingRate(  
    MINSTANCE mInst, double *ratio);
```

N/A

```
<font color="green">// Get the current threading rate from the core.</font>  
MINSTANCE mInst = 0; double ratio; int rc =  
mcMotionGetThreadingRate(mInst, &ratio);
```

```
int mcMotionGetVel(MINSTANCE mInst, int motorId, double *velocity);  
velocity, rc = mc.mcMotionGetVel(  
    number mInst, number motorId)  
  
<font color="green">// Get the motor velocity for motor 1</font>  
MINSTANCE mInst = 0;  
  
double vel;  
  
int rc = mcMotionGetVel(mInst, 1, &vel);
```

```
int mcMotionSetCycleTime(  
    MINSTANCE mInst, double secs);
```

N/A

```
<font color="green">// Set the cycle time slice.</font> MINSTANCE mInst  
= 0; int mcMotionSetCycleTime(mInst, .001);
```

```
int mcMotionSetMoveID(  
    MINSTANCE mInst, int id);
```

N/A

// Set the currently executing movement ID.
MINSTANCE mInst = 0; int moveId = 10001; //
Should be obtained from the motion controller. int
mcMotionSetMoveID(mInst, moveId);

```
int mcMotionSetPos(
```

```
    MINSTANCE mInst, int motorId double val);
```

N/A

```
<font color="green">// Set the motor position for motor 1.</font>
MINSTANCE mInst = 0; motorCounts = 324255; <font color="green">//
Should be retrieved from the motion controller.</font> int
mcMotionSetPos(mInst, 1, motorCounts);
```

```
int mcMotionSetProbeComplete(
```

```
    MINSTANCE mInst);
```

N/A

```
<font color="green">// Complete a probe operation.</font> MINSTANCE  
mInst = 0; int rc = mcMotionSetProbeComplete(mInst);
```

```
int mcMotionSetProbePos(
```

```
    MINSTANCE mInst, int motorId, double val);
```

N/A

// Set the probed position for motor 0.

```
MINSTANCE mInst = 0; double probedPos = 2314134; <font
```

color="green">// Should be reteived from motion controller latch registers.

```
</font> int rc = mcMotionSetProbePos(mInst, 0, double val);
```

```
int mcMotionSetStill(  
    MINSTANCE mInst, int motorId);
```

N/A

```
<font color="green">// Motor stop report example.</font> MINSTANCE  
mInst = 0; execution_t ex;  
  
mcMotionCyclePlannerEx(m_cid, &ex); switch(ex.exType) {  
  
case EX_STOP_REQ:  
  
    // See if we need to report when the motors are still.  
  
    for (i = 0; i < 8; i++) {  
  
        if (ex.exMotors[i].reportStopped == TRUE) {  
  
            <font color="green">// Report when this motor has completed all  
previous moves!</font> }  
  
    }  
  
    break; ...  
}
```

```
int mcMotionSetThreadingRate(  
    MINSTANCE mInst, double ratio);
```

N/A

```
<font color="green">// Set the threading ratio.</font> MINSTANCE mInst  
= 0; int rc = mcMotionSetThreadingRate(mInst, 1.2);
```

```
int mcMotionSetVel(  
    MINSTANCE mInst, int motorId, double velocity);
```

N/A

```
<font color="green">// Report the velocity for motor 0.</font>  
MINSTANCE mInst = 0; double vel = 2342420;<font color="green">//  
Should be obtained from the motion controller.</font> int  
mcMotionSetVel(mInst, 0, vel);
```

```
int mcMotionSync(
```

```
    MINSTANCE mInst);
```

N/A

```
<font color="green">// Synch core planner positions with the last reported  
motor positions.</font> MINSTANCE mInst = 0; int rc =  
mcMotionSync(mInst);
```

```
int mcMotionThreadComplete(
```

```
    MINSTANCE mInst);
```

N/A

```
<font color="green">// Report that the threading op is complete.</font>
MINSTANCE mInst = 0; int rc = mcMotionThreadComplete(mInst);
```

```
int mcMotorGetAxis(  
    MINSTANCE mInst, int motorId, int *axisId)  
  
axisId, rc = mc.mcMotorGetAxis(  
    number mInst, number motorId)  
  
<font color="green">// Get the parent axis for motor 0.</font>  
MINSTANCE mInst = 0; int axisId = -1;  
  
int rc = mcMotorGetAxis(mInst, 0, &axisId);
```

```
int mcMotorGetCountsPerUnit(  
    MINSTANCE mInst, int motorId, double *counts)  
counts, rc = mc.mcMotorGetCountsPerUnit(  
    number mInst, number motorId)  
  
<font color="green">// Get the counts per unit for motor 0.</font>  
MINSTANCE mInst = 0;  
  
double counts = 0.0;  
  
int rc = mcMotorGetCountsPerUnit(mInst, 0, &counts);
```

```
mcMotorGetInfoStruct(
```

```
    MINSTANCE mInst, int motorId,
```

```
    motorinfo_t *minf);
```

N/A

```
struct motorinfo {
```

```
    double CountsPerUnit; // Number of encoder counts or steps per unit.
```

```
    double MaxVelocity; // Max velocity of the axis.
```

```
    double MaxAcceleration; // Max rate to accelerate.
```

```
    BOOL Reverse; // Is the axis reversed?
```

```
    double BacklashAmount; // The amount of backlash in counts.
```

```
    double CurrentVelocity; // The speed the axis is moving, This could be  
    reported by the motion deivce.
```

```
    int CurrentPosition; // The Current Position (From the motion device).
```

```
    BOOL Homed; // True if the axis has been homed.
```

```
    long SoftMaxLimit; // Count for the max travel.
```

```
    long SoftMinLimit; // Count for the min travel.
```

```
    BOOL CanHome; // Can this motor home?
```

```
    BOOL Enabled; // Is this motor enabled?
```

```
    long EnableDelay; // ms to delay the enable signal for this motor.
```

```
    int AxisId; // -1 if no axis has mapped this motor.
```

```
};

typedef struct motorinfo motorinfo_t;

MINSTANCE mInst = 0;

int m = 2;

motorinfo_t minf;

mcMotorGetInfoStruct(mInst, m, &minf); <font color="green">// Get data
for motor 2.</font>
```

```
int mcMotorGetMaxAccel(  
    MINSTANCE mInst, int motorId, double *maxAccel);  
  
maxAccel, rc = mc.mcMotorGetMaxAccel(  
    number mInst, number motorId)  
  
<font color="green">// Get the max accel for motor 0.</font>  
MINSTANCE mInst = 0;  
  
double maxAccel = 0.0;  
  
int rc = mcMotorGetMaxAccel(mInst, 0, &maxAccel);
```

```
int mcMotorGetMaxVel(  
    MINSTANCE mInst, int motorId, double *maxVel)  
  
maxVel, rc = mc.mcMotorGetMaxVel(  
    number mInst, number motorId)  
  
<font color="green">// Get the max vel for motor 0.</font> MINSTANCE  
mInst = 0;  
  
double maxVel = 0.0;  
  
int rc = mcMotorGetMaxVel(mInst, 0, &maxVel);
```

```
int mcMotorGetPos(  
    MINSTANCE mInst, int motorId, double *val);  
  
val, rc = mc.mcMotorGetPos(  
    number mInst, number motorId)  
  
MINSTANCE mInst = 0; double Motor1Pos = 0; <font color="green">//  
Get the position of the Motor 1.</font> mcMotorGetPos(mInst, 1,  
&Motor1Pos);
```

```
int mcMotorGetVel(  
    MINSTANCE mInst, int motor, double *velocity);  
  
velocity, rc = mc.mcMotorGetVel(  
    number mInst, number motor)  
  
MINSTANCE mInst = 0; int m = MOTOR2;  
  
double CurrentVel = 0; mcMotorSetVel(mInst, m, &CurrentVel); <font  
color="green">// Get the current speed of motor2.</font>
```

```
int mcMotorIsHomed(  
    MINSTANCE mInst, int motorId, BOOL *homed);  
  
homed, rc = mc.mcMotorIsHomed(  
    number mInst, number motorId)  
  
MINSTANCE mInst = 0; int m = MOTOR2;  
  
int Homed = 0;  
  
mcMotorIsHomed(mInst, m, &Homed); <font color="green">// Get the  
homed state of motor2.</font>
```

```
int mcMotorIsStill(  
    MINSTANCE mInst, int motorId, BOOL *still);  
  
still, rc = mc.mcMotorIsStill(  
    number mInst, number motorId)  
  
MINSTANCE mInst = 0; int m = MOTOR0;  
  
int still = 0;  
  
mcMotorSetStill(mInst, m, &still); <font color="green">// Get the state if  
MOTOR0 is still</font>
```

```
int mcMotorMapGetDefinition(  
    MINSTANCE mInst, int motorId, long *lengthCounts, long *numPoints);  
  
lengthCounts, numPoints, rc = mc.mcMotorMapGetDefinition(  
    number mInst, number motorId)  
  
<font color="green">// </font> MINSTANCE mInst = 0;  
  
long lengthCounts;  
  
long numPoints;  
  
int rc = mcMotorMapGetDefinition(mInst, 0, &lengthCounts,  
    &numPoints);
```

```
int mcMotorMapGetLength(  
    MINSTANCE mInst, int motorId, int *length);  
  
length, rc = mcMotorMapGetLength(  
    number mInst, number motorId)  
  
<font color="green">// Get the screw length for motor zero.</font>  
MINSTANCE mInst = 0;  
  
int length;  
  
int rc = mcMotorMapGetLength(mInst, 0, &length);
```

```

int mcMotorMapGetNPoints(
    MINSTANCE mInst, int motorId, int *points);

points, rc = mc.mcMotorMapGetNPoints(
    number mInst, number motorId)

```

Description: Retrieve the number of measurement points in the screw map for the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
points	the address of an integer to receive the number of measurement points.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by motorId was not found.
MERROR_INVALID_ARG	points is NULL.

Notes:

Deprecated. Use mcMotorMapGetPointCount() instead.

```
int mcMotorMapGetPoint(  
    MINSTANCE mInst, int motorId, int point, int *error);  
  
error, rc = mc.mcMotorMapGetPoint(  
    numbser mInst, numbser motorId, numbser point)  
  
<font color="green">// Get the screw map error for motor 0, measurement  
point 10.</font> MINSTANCE mInst = 0;  
  
int screwErr;  
  
int rc = mcMotorMapGetPoint(mInst, 0, 10, &screwErr);
```

```
int mcMotorMapGetPointCount(  
    MINSTANCE mInst, int motorId, int *points);  
  
points, rc = mc.mcMotorMapGetPointCount(  
    number mInst, number motorId)  
  
<font color="green">// Get the number of measurement points.</font>  
MINSTANCE mInst = 0;  
  
int points;  
  
int rc = mcMotorMapGetPointCount(mInst, 0, &points);
```

```
int mcMotorMapGetStart(  
    MINSTANCE mInst, int motorId, int *startPoint);  
  
startPoint, rc = mc.mcMotorMapGetStart(  
    number mInst, number motorId)  
  
<font color="green">// Get the starting point for the motor 0 screw map.  
</font> MINSTANCE mInst = 0;  
  
int startPoint;  
  
int rc = mcMotorMapGetStart(mInst, 0, &startPoint);
```

```
int mcMotorMapSetDefinition(  
    MINSTANCE mInst, int motorId, long lengthCounts, long numPoints);  
  
rc = mc.mcMotorMapSetDefinition(  
    number mInst, number motorId, number lengthCounts, number  
    numPoints)  
  
<font color="green">// Set the screw map definition for motor 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetDefinition(mInst, 0, 20000, 20);
```

```
int mcMotorMapSetLength(  
    MINSTANCE mInst, int motorId, int length);  
  
rc = mc.mcMotorMapSetLength(  
    number mInst, number motorId, number length);  
  
<font color="green">// Set the screw length for the motor 0 screw map.  
</font> MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetLength(mInst, 0, 20000);
```

```

int mcMotorMapSetNPoints(
    MINSTANCE mInst, int motorId, int points);

rc = mc.mcMotorMapSetNPoints(
    number mInst, number motorId, number points);

```

Description: Set the number of measured points for the screw map of the given motor.

Parameters:

Parameter	Description
mInst	The controller instance.
motorId	The motor id.
points	An integer specifying the number of measured points.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.
MERROR_MOTOR_NOT_FOUND	The motor specified by motorId was not found.
MERROR_INVALID_ARG	points is ≤ 0 .

Notes:

Deprecated. Use mcMotorMapSetPointCount() instead.

```
int mcMotorMapSetPoint(  
    MINSTANCE mInst, int motorId, int point, int error);  
  
rc = mc.mcMotorMapSetPoint(  
    number mInst, number motorId, number point, number error);  
  
<font color="green">// Set the error in motor 0 screw map.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetPoint(mInst, 0, 5, 12);
```

```
int mcMotorMapSetPointCount(  
    MINSTANCE mInst, int motorId, int points);  
  
rc = mc.mcMotorMapSetPointCount(  
    number mInst, number motorId, number points);  
  
<font color="green">// Set the number of measured points in the screw map  
for motor 0.</font> MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetPointCount(mInst, 0, 20);
```

```
int mcMotorMapSetStart(  
    MINSTANCE mInst, int motorId, int start);  
  
rc = mc.mcMotorMapSetStart(  
    number mInst, number motorId, number start)  
  
<font color="green">// Set the starting point number for the motor 0 screw  
map. (usually 0)</font> MINSTANCE mInst = 0;  
  
int rc = mcMotorMapSetStart(mInst, 0, 0);
```

mcMotorRegister(

 MINSTANCE mInst, int motorId);

N/A

 MINSTANCE mInst = 0; for (int m = MOTOR0; m < MOTOR4; m++) {

 mcMotorRegister(mInst, m);// Register motor0 -
 motor2 in the core. }

```
int mcMotorSetCountsPerUnit(  
    MINSTANCE mInst, int motorId, double counts);  
  
rc = mc.mcMotorSetCountsPerUnit(  
    number mInst, number motorId, number counts)  
  
<font color="green">// Set the counts for motor 0.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcMotorSetCountsPerUnit(mInst, 0, 1024);
```

```
int mcMotorSetHomePos(  
    MINSTANCE mInst, int motorId, int count);  
  
rc = mc.mcMotorSetHomePos(  
    number mInst, number motorId, number count)  
    MINSTANCE mInst = 0;  
  
int m = MOTOR2;  
  
int count = 1200;  
  
mcMotorSetHomePos(mInst, m, count); <font color="green">// Set the  
Home position of motor2.</font>
```

```
int mcMotorSetInfoStruct(  
    MINSTANCE mInst, int motoId,  
    motorinfo_t *minf);
```

N/A

```
struct motorinfo {  
    double CountsPerUnit; // Number of encoder counts or steps per unit.  
    double MaxVelocity; // Max velocity of the axis.  
    double MaxAcceleration; // Max rate to accelerate.  
    BOOL Reverse; // Is the axis reversed?  
    double BacklashAmount; // The amount of backlash in counts.  
    double CurrentVelocity; // The speed the axis is moving, This could be  
    // reported by the motion device.  
    int CurrentPosition; // The Current Position (From the motion device).  
    BOOL Homed; // True if the axis has been homed.  
    long SoftMaxLimit; // Count for the max travel.  
    long SoftMinLimit; // Count for the min travel.  
    BOOL CanHome; // Can this motor home?  
    BOOL Enabled; // Is this motor enabled?  
    long EnableDelay; // ms to delay the enable signal for this motor.  
};
```

```
typedef struct motorinfo motorinfo_t;

MINSTANCE mInst = 0;

int m = MOTOR2;

motorinfo_t minf;

mcMotorGetInfoStruct(mInst, m, &minf); <font color="green">// Get data
for motor2.</font> minf.CountsPerUnit /= 2;<font color="green">//Divid
motor counts per unit by 2.</font> mcMotorSetInfoStruct(mInst, m,
&minf); <font color="green">// Set data for motor2.</font>
```

```
int mcMotorSetMaxAccel(  
    MINSTANCE mInst, int motorId, double maxAccel);  
  
rc = mc.mcMotorSetMaxAccel(  
    number mInst, number motorId, number maxAccel)  
  
<font color="green">// Set a new accel value for motor 0.</font>  
MINSTANCE mInst = 0;  
  
int mcMotorSetMaxAccel(mInst, 0, 75);
```

```
int mcMotorSetMaxVel(  
    MINSTANCE mInst, int motorId, double maxVel);  
  
rc = mc.mcMotorSetMaxVel(  
    number mInst, number motorId, number maxVel)  
  
<font color="green">// Set the maximum velocity for motor 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMotorSetMaxVel(mInst, 0, 500);
```

```
int mcMotorUnregister(
```

```
    MINSTANCE mInst, int motorId);
```

N/A

```
MINSTANCE mInst = 0; int m = MOTOR2; mcMotorUnregister(mInst,  
m);<font color="green">// Unregister motor2 from the Core.</font>
```

```
int mcMpgGetAccel(  
    MINSTANCE mInst, int mpg, double *percentMaxAccel);  
  
percentMaxAccel, rc = mc.mcMpgGetAccel(  
    number mInst, number mpg)  
  
<font color="green">// Get the current accel for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
double percentMaxAccel = 0; int rc;  
  
rc = mcMpgGetAccel(mInst, 0, &percentMaxAccel);
```

```
int mcMpgGetAxis(  
    MINSTANCE mInst, int mpg, int *axisId);  
  
axisId, rc = mc.mcMpgGetAxis(  
    number mInst, number mpg)  
  
<font color="green">// Get the currently selected axis for MPG 0.</font>  
MINSTANCE mInst = 0; int axisId;  
  
int rc = mcMpgGetAxis(mInst, 0, &axisId);
```

```
int mcMpgGetCountsPerDetent(  
    MINSTANCE mInst, int mpg, int *pulses);  
  
pulses, rc = mc.mcMpgGetCountsPerDetent(  
    number mInst, number mpg)  
  
<font color="green">// Get the counts per detent for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
int cnts;  
  
int rc = mcMpgGetCountsPerDetent(mInst, 0, &cnts);
```

```
int mcMpgGetEnable(  
    MINSTANCE mInst, int mpg, BOOL *enabled);  
  
enabled, rc = mc.mcMpgGetEnable(  
    number mInst, number mpg)  
  
<font color="green">// Get the enabled status of MPG 0.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = MC_FALSE;  
  
  
  
int rc = mcMpgGetEnable(mInst, 0, &enabled);
```

```
int mcMpgGetEncoderReg(  
    MINSTANCE mInst, int mpg, HMCREG *hReg);  
  
hReg, rc = mc.mcMpgGetEncoderReg(  
    number mInst, number mpg)  
  
<font color="green">// Get the current encoder associated with MPG 0.  
</font> MINSTANCE mInst = 0;  
  
HMCREG hReg = 0;
```

```
int rc = mcMpgGetEncoderReg(mInst, 0, &hReg);
```

```
int mcMpgGetInc(  
    MINSTANCE mInst, int mpg, double *inc);  
  
inc, rc = mc.mcMpgGetInc(  
    number mInst, number mpg)  
  
<font color="green">// Get the incremnt for MPG 0.</font> MINSTANCE  
mInst = 0; double inc;  
  
int rc = mcMpgGetInc(mInst, 0, &inc);
```

```
int mcMpgGetRate(  
    MINSTANCE mInst, int mpg, double *percentMaxVel);  
  
percentMaxVel, rc = mc.mcMpgGetRate(  
    number mInst, number mpg)  
  
<font color="green">// Get the velocity percentage for MPG 0.</font>  
MINSTANCE mInst = 0; double velPercent;  
  
int rc = mcMpgGetRate(mInst, 0, &velPercent);
```

```
int mcMpgGetReversed(  
    MINSTANCE mInst, int mpg, BOOL *reversed);  
  
reversed, rc = mc.mcMpgGetReversed(  
    number mInst, number mpg)  
  
<font color="green">// Get the reversed status of MPG 0.</font>  
MINSTANCE mInst = 0;  
  
BOOL reversed = MC_FALSE;  
  
int rc = mcMpgGetReversed(mInst, 0, &reversed);
```

```
int mcMpgGetShuttleMode(  
    MINSTANCE mInst, BOOL *on);  
  
on, rc = mc.mcMpgGetShuttleMode(  
    number mInst)  
  
<font color="green">// Check the state of MPG shuttle mode.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcMpgGetShuttleMode(mInst, &enabled);
```

```
int mcMpgGetShuttlePercent(  
    MINSTANCE mInst, int mpg, double *percent);  
  
percent, rc = mc.mcMpgGetShuttlePercent(  
    number mInst, number mpg)  
  
<font color="green">// Get the shuttle percentage for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
double percent;  
  
int rc = mcMpgGetShuttlePercentage(mInst, 0, &percent);
```

```
int mcMpgMoveCounts(  
    MINSTANCE mInst, int mpg, int deltaCounts);  
  
rc = mc.mcMpgMoveCounts(  
    number mInst, number mpg, number deltaCounts)  
  
<font color="green">// Move the MPG 0 "n" counts.</font> MINSTANCE  
mInst = 0;  
  
int n = 16;  
  
int rc = mcMpgMoveCounts(mInst, 0, n);
```

```
int mcMpgSetAccel(  
    MINSTANCE mInst, int mpg, double percentMaxAccel);  
  
rc = mc.mcMpgSetAccel(  
    number mInst, number mpg, number percentMaxAccel)  
  
<font color="green">// Set the acceleration value for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMpgSetAccel(mInst, 0, 20.5);
```

```
int mcMpgSetAxis(
```

```
    MINSTANCE mInst, int mpg, int axisId);
```

```
    rc = mc.mcMpgSetAxis(
```

```
        number mInst, number mpg, number axis)
```

```
<font color="green">// Set MPG 0 to move the X axis.</font>
```

```
    MINSTANCE mInst = 0; int rc = mcMpgSetAxis(mInst, 0, X_AXIS);
```

```
int mcMpgSetCountsPerDetent(  
    MINSTANCE mInst, int mpg, int pulses);  
  
rc = mc.mcMpgSetCountsPerDetent(  
    number mInst, number mpg, number pulses);  
  
<font color="green">// Set the number of counts per detent for MPG 0.  
</font> MINSTANCE mInst = 0;  
  
int rc = mcMpgSetCountsPerDetent(mInst, 0, 4);
```

```
int mcMpgSetEnable(  
    MINSTANCE mInst, int mpg, BOOL enabled);  
  
rc = mc.mcMpgSetEnable(  
    number mInst, number mpg number enabled)  
  
<font color="green">// Set the enabled status of MPG 0.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = MC_FALSE;  
  
int rc = mcMpgSetEnable(mInst, 0, MC_TRUE);
```

```
int mcMpgSetEncoderReg(  
    MINSTANCE mInst, int mpg, HMCREG hReg);  
  
rc = mc.mcMpgSetEncoderReg(  
    number mInst, number mpg number hReg)  
  
<font color="green">// Associate an encoder with MPG 0.</font>  
MINSTANCE mInst = 0;  
  
HMCREG hReg = 0;
```

```
int rc  
  
rc = mcRegGetHandle(mInst, "/Sim0/Encoder0", &hReg); if (rc ==  
MERROR_NOERROR) {  
  
    rc = mcMpgSetEncoderReg(mInst, 0, hReg); }  
  
}
```

```
int mcMpgSetInc(  
    MINSTANCE mInst, int mpg, double inc);  
  
rc = mc.mcMpgSetInc(  
    number mInst, number mpg, number inc);  
  
<font color="green">// Set the increment to .001 for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMpgSetInc(mInst, 0, .001);
```

```
int mcMpgSetRate(  
    MINSTANCE mInst, int mpg, double percentMaxVel);  
  
rc = mc.mcMpgSetRate(  
    number mInst, number mpg, number percentMaxVel)  
  
<font color="green">// Set the rate for MPG 0 to 25.0%</font>  
MINSTANCE mInst = 0;  
  
int rc = mcMpgSetRate(mInst, 0, 25.0);
```

```
int mcMpgSetReversed(  
    MINSTANCE mInst, int mpg, BOOL reversed);  
  
rc = mc.mcMpgGetReversed(  
    number mInst, number mpg number reversed)  
  
<font color="green">// Get the reversed status of MPG 0.</font>  
MINSTANCE mInst = 0;  
  
BOOL reversed = MC_FALSE;  
  
int rc = mcMpgSetReversed(mInst, 0, reversed); <font color="green">//  
MC_TRUE</font>
```

```
int mcMpgSetShuttleMode(  
    MINSTANCE mInst, BOOL on);  
  
rc = mc.mcMpgSetShuttleMode(  
    number mInst, number on);  
  
<font color="green">// Turn on MPG shuttle mode.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcMpgSetShuttleMode(mInst, MC_ON);
```

```
int mcMpgSetShuttlePercent(  
    MINSTANCE mInst, int mpg, double percent);  
  
rc = mc.mcMpgGetShuttlePercent(  
    number mInst, number mpg number percent)  
  
<font color="green">// Set the shuttle percentage for MPG 0.</font>  
MINSTANCE mInst = 0;  
  
double percent = 200.5;  
  
int rc = mcMpgGetShuttlePercentage(mInst, 0, percent);
```

```
int mcPluginConfigure(  
    MINSTANCE mInst, int plugId);
```

N/A

```
MINSTANCE mInst = 0;  
HMCPLUG hPlug = 0;  
plugininfo_t pluginf;
```

```
<font color="green">// Loop through all of the plugins looking for  
plugins<br/>// that have configuration dialogs.</font> while  
(mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug) ==  
MERROR_NOERROR) {  
  
    <font color="green">// Get the plugin info that contains the plugin ID.  
</font> rc = mcPluginGetInfoStruct(hPlug, &pluginf); if (rc ==  
MERROR_NOERROR) {  
  
    int pluginId = pluginf.plugId; <font color="green">// Call the  
configuration dialog.</font> rc = <font  
color="blue">mcPluginConfigure(mInst, pluginId);</font>; }  
  
}
```

```
int mcPluginCoreLoad(
```

```
    const char *shortName);
```

N/A

```
<font color="green">// Load the mcGalil plugin.</font> int rc =  
mcPluginCoreLoad("mcGalil");
```

```
int mcPluginCoreUnload(
```

```
    const char *shortName);
```

N/A

```
<font color="green">// Unload the mcGalil plugin.</font> int rc =  
mcPluginCoreUnload("mcGalil");
```

```
int mcPluginDiagnostic(  
    MINSTANCE mInst, int pluginId);
```

N/A

```
MINSTANCE mInst = 0;
```

```
HMCPLUG hPlug = 0;
```

```
plugininfo_t plugininf;
```

```
<font color="green">// Loop through all of the plugins looking for  
plugins<br/>// that have diagnostic dialogs.</font> while  
(mcPluginGetNextHandle(PLUG_TYPE_DIAG, hPlug, &hPlug) ==  
MERROR_NOERROR) {
```

```
    <font color="green">// Get the plugin info that contains the plugin ID.  
</font> rc = mcPluginGetInfoStruct(hPlug, &plugininf); if (rc ==  
MERROR_NOERROR) {
```

```
        int pluginId = plugininf.pluginId; <font color="green">// Call the diagnostics  
dialog.</font> rc = <font color="blue">mcPluginDiagnostic(mInst,  
pluginId);</font>; }
```

```
}
```

```
int mcPluginEnable(  
    HMCPLUG hPlug, BOOL enable);
```

N/A

```
HMCPLUG hPlug = 0; // Loop through all of the  
plugins looking for plugins<br/>// that have configuration dialogs.</font>  
while (mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug)  
== MERROR_NOERROR) {  
  
    // Enable the plugin.</font> rc = <font  
    color="blue">mcPluginEnable(hPlug, TRUE)</font>; }
```

```
int mcPluginGetEnabled(
```

```
    HMCPLUG hPlug, BOOL *enabled);
```

N/A

```
HMCPLUG hPlug = 0;
```

```
BOOL enabled = FALSE;
```

```
<font color="green">// Loop through all of the plugins looking for  
plugins<br/>// that have configuration dialogs.</font> while  
(mcPluginGetNextHandle(PLUG_TYPE_CONFIG, hPlug, &hPlug) ==  
MERROR_NOERROR) {
```

```
    <font color="green">// See if the plugin is enabled..</font> rc = <font  
color="blue">mcPluginGetEnabled(hPlug, &enabled)</font>; if (rc ==  
MERROR_NOERROR) {
```

```
        if (enabled == TRUE) {
```

```
            <font color="green">// The plugin is enabled!!!</font> } else {
```

```
            <font color="green">// The plugin is not enabled!!!</font> }
```

```
}
```

```
}
```

```
mcPluginGetInfoStruct(HMCPLUG hPlug, pluginfo_t *pluginf);
```

N/A

```
HMCPLUG hPlug = 0; pluginfo_t pluginf;
```

```
<font color="green">// Loop through all of the plugins.
```

```
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug) ==  
MERROR_NOERROR) {
```

```
    <font color="green">// Get the plugin info that contains the plugin ID.  
</font> rc = <font color="blue">mcPluginGetInfoStruct(hPlug, &pluginf)  
</font>; if (rc == MERROR_NOERROR) {
```

```
        int pluginId = pluginf.pluginId; ...
```

```
}
```

```
}
```

```
</font>
```

```
int mcPluginGetLicenseFeature(
```

```
    HMCPLUG hPlug, int index, char *buf, int bufSize, BOOL *status);
```

N/A

```
HMCPLUG hPlug = 0;
```

```
<font color="green">// Loop through all of the plugins.
```

```
while (mcPluginGetNextHandle(PLUG_TYPE_ALL, hPlug, &hPlug) ==  
MERROR_NOERROR) {
```

```
    <font color="green">// Get all of the plugin license features</font> int  
    index = 0; char buf[80]; BOOL licensed = FALSE; while (<font  
    color="blue">mcPluginGetLicenseFeature(hPlug, index, buf, sizeof(buf),  
    &licensed)</font> == MERROR_NOERROR) {
```

```
        printf("license feature %s is %s.\n", buf, licensed == TRUE ? "licensed" :  
        "not licensed"); index++; }
```

```
}
```

```
</font>
```

```
int mcPluginGetNextHandle(
```

```
    int plugType, HMCPLUG startPlug, HMCPLUG *hPlug);
```

N/A

```
HMCPLUG hPlug = 0;
```

```
<font color="green">// Loop through all of the plugins looking for  
plugins<br/>// that have configuration dialogs and provide I/O.</font> int  
attr = PLUG_TYPE_CONFIG | PLUG_TYPE_IO; while (<font  
color="blue">mcPluginGetNextHandle(attr, hPlug, &hPlug)</font> ==  
MERROR_NOERROR) {
```

```
    <font color="green">// Enable the plugin.</font> rc =  
    mcPluginEnable(hPlug, TRUE); }
```

```
int mcPluginGetValid(
```

```
    HMCPLUG hPlug, BOOL *valid);
```

N/A

```
HMCPLUG hPlug = 0;
```

```
BOOL valid = FALSE;
```

```
<font color="green">// Loop through all of the plugins checking to see if  
they are valid.</font> while (mcPluginGetNextHandle(PLUG_TYPE_ALL,  
hPlug, &hPlug) == MERROR_NOERROR) {
```

```
    <font color="green">// Check the plugin validity.</font> rc = <font  
color="blue">mcPluginGetValid(hPlug, &valid)</font>; if (rc ==  
MERROR_NOERROR) {
```

```
        if (valid == TRUE) {
```

```
            <font color="green">// The plugin is valid and will function.</font> }  
        else {
```

```
            <font color="green">// The plugin is invalid and will not function.  
</font> }
```

```
    }
```

```
}
```

```
int mcPluginInstall(  
    const char *m4plug);
```

N/A

```
<font color="green">// Install the Windows mcGalil plugin from an  
installation archive.</font> int rc = mcPluginInstall("mcGalil.m4Plugw");  
if (rc == MERROR_NOERROR) {  
  
<font color="green">//The plugin installed successfully.</font> }
```

```
int mcPluginRegister(
```

```
    HMCPLUG hPlug, const char *DeveloperId, const char *Desc, const
    char *Version, int Type);
```

N/A

```
<font color="green">// This function gets called (only once) when the
Plugin is loaded by Mach Core.</font> MCP_API int MCP_APIENTRY
mcPluginLoad(HMCPLUG id) {
```

...

```
<font color="blue">mcPluginRegister(id, "Newfangled", "Simulator -
Newfangled Solutions", MC_VERSION_STR, PLUG_TYPE_MOTION |
PLUG_TYPE_IO | PLUG_TYPE_REG | PLUG_TYPE_CONFIG |
PLUG_TYPE_DIAG)</font>; return(MERROR_NOERROR); }
```

```
int mcPluginRemove(const char *shortName);
```

N/A

```
<font color="green">// Remove the mcGalil plugin.</font> MINSTANCE  
mInst = 0; int rc = mcPluginRemove("mcGalil");
```

```
int mcProfileDeleteKey(  
    MINSTANCE mInst, const char *section, const char *key);  
  
rc = mc.mcProfileDeleteKey(  
    number mInst, string section, string key)  
  
MINSTANCE mInst = 0;  
  
mcProfileDeleteKey(mInst , "P_Port", "Frequency");
```

```
int mcProfileDeleteSection(  
    MINSTANCE mInst, const char *section);  
  
rc = mc.mcProfileDeleteSection(  
    number mInst, string section)  
  
MINSTANCE mInst = 0;  
  
mcProfileDeleteKey(mInst , "P_Port");
```

```
int mcProfileExists(  
    MINSTANCE mInst, const char *section, const char *key);  
  
rc = mc.mcProfileExists(  
    number mInst, string section, string key)  
  
MINSTANCE mInst = 0;  
  
mcProfileWriteInt(mInst, "P_Port", "Frequency");
```

```
int mcProfileFlush(
```

```
    INSTANCE mInst);
```

```
rc = mc.mcProfileFlush(
```

```
    number mInst)
```

```
MINSTANCE mInst = 0; mcProfileFlush(mInst); <font color="green">//  
Flush changes to the INI file.</font>
```

```
mcProfileGetDouble(
```

```
    MINSTANCE mInst, const char *section, const char *key, double  
    *retval, double defval);
```

```
retval, rc = mc.mcProfileGetDouble(
```

```
    number mInst, string section, string key, number defval)
```

```
double rval=0;
```

```
MINSTANCE mInst = 0;
```

```
mcProfileGetDouble(mInst , "P_Port", "Frequency", &rval, 25.34);
```

```
int mcProfileGetInt(  
    MINSTANCE mInst, const char *section, const char *key, long *retval,  
    long defval);  
  
retval, rc = mc.mcProfileGetInt(  
    number mInst, string section, string key, number defval)  
  
long rval=0;  
  
int mInst=0;  
  
mcProfileGetInt(mInst , "P_Port", "Frequency", &rval, 25000);
```

```
int mcProfileGetName(  
    MINSTANCE mInst, char *buff, size_t bufsize);  
  
name, rc = mc.mcProfileGetName(  
    number mInst)  
  
MINSTANCE mInst = 0; char buff[80];  
  
memset(buff, 0, 80); mcProfileGetName(mInst, buff, 80);
```

```
int mcProfileGetString(  
    MINSTANCE mInst, const char *section, const char *key, char *buff,  
    long bufsize, const char *defval);  
  
buff, rc = mc.mcProfileGetString(  
    MINSTANCE mInst, string section, string key, string defval);  
  
MINSTANCE m_inst = 0;  
  
char *key = "BufferedTime"; char buff[80];  
  
memset(buff, 0, 80);  
  
mcProfileGetString(mInst , "SomeSection", key, buff, 80, "0.100");
```

```
int mcProfileReload(  
    INSTANCE mInst);  
  
rc = mc.mcProfileReload(  
    number mInst)  
  
MINSTANCE mInst = 0; mcProfileReload(mInst); <font color="green">//  
Reload the settings from the INI file.</font>
```

```
int mcProfileSave(  
    INSTANCE mInst);  
  
rc = mc.mcProfileSave(  
    number mInst)
```

```
MINSTANCE mInst = 0; mcProfileSave(mInst); <font color="green">//  
Flush the settings to the INI file.</font>
```

```
int mcProfileWriteDouble(  
    MINSTANCE mInst, const char *section, const char *key, double val);  
  
rc = mc.mcProfileWriteDouble(  
    number mInst, string section, string key, double val)  
  
MINSTANCE mInst=0;  
  
mcProfileWriteDouble(m_cid , "P_Port", "Frequency", 45000);
```

```
int mcProfileWriteInt(  
    MINSTANCE mInst, const char *section, const char *key, long val);  
  
int mcProfileWriteInt(  
    number mInst, string section, string key, number val)  
  
MINSTANCE mInst = 0;  
  
mcProfileWriteInt(m_cid , "P_Port", "Frequency", 45000);
```

```
int mcProfileWriteString(  
    MINSTANCE mInst, const char *section, const char *key, const char  
    *val);  
  
rc = mc.mcProfileWriteString(  
    number mInst, string section, string key, string val)  
  
MINSTANCE m_inst = 0;  
  
char *key = "BufferedTime"  
  
double BuffTime = .250;  
  
char val[80];  
  
sprintf(val, "%.4f", BuffTime); mcProfileWriteString(mInst, "Darwin", key,  
val);
```

```
int mcRegGetCommand(
```

```
    HMCREG hReg,
```

```
    char *cmd,
```

```
    size_t cmdLen)
```

N/A

```
<font color="green">// Retrieve the register command.</font> int  
simControl::ProcessMsg(long msg, long param1, long param2) {
```

```
    HMCREG hReg = (HMCREG)param1; long RegVal;
```

```
    switch (msg) {
```

```
        case MSG_REG_COMMAND: mcRegGetValueLong(hReg, &RegVal);  
        if (hReg = m_RegCommand) { <font color="green">// Is this our command  
        register?</font> char command[1024]; mcRegGetCommand(hReg,  
        command, sizeof(command)); wxString cmd(command);  
        cmd.MakeUpper(); if (cmd == wxT("THC ON")) {
```

```
            m_thc = true; mcRegSetResponse(hReg, "OK"); } else if (cmd ==  
            wxT("THC OFF")) {
```

```
            m_thc = false; mcMotionSync(m_cid); mcRegSetResponse(hReg, "OK");  
        } else if (cmd == wxT("THC STATUS")) {
```

```
            if (m_thc) {
```

```
                mcRegSetResponse(hReg, "1"); } else {
```

```
                mcRegSetResponse(hReg, "0"); }
```

```
        }
```

```
}
```

```
break;  
default:  
;  
}  
return(MERROR_NOERROR); }
```

```
int mcRegGetHandle(  
    MINSTANCE mInst, const char *path, HMCREG *hReg);  
  
hReg, rc = mc.mcRegGetHandle(  
    number mInst, string path)  
  
<font color="green">// Get the register handle.</font> MINSTANCE mInst  
= 0; HMCREG hReg;  
  
int rc;  
  
double value;  
  
if (<font color="blue">mcRegGetHandle(mInst, "core/global/Version",  
&hReg)</font> == MERROR_NOERROR) {  
  
    rc = mcRegGetValue(hReg, &value); }  
 
```

```
int mcRegGetInfo(
    HMCREG hReg, char *nameBuf, size_t nameBuflen, char *descBuf,
    size_t descBuflen, int *type, HMCDEV *hDev);

nameBuf, descBuf, type, hDev, rc = mc.mcRegGetInfo(
    number hReg)

HMCREG hReg = 0; char name[80];
char desc[80];
int type;
HMCDEV hDev;

while (mcSignalGetNextHandle(hDev, hReg, &hReg) ==
MERROR_NOERROR) {

    if (hSig != 0) {

        <font color="green">// Get the info on the register.</font>
mcRegGetInfo(hReg, name, sizeof(name), desc, sizeof(desc), &type,
&hDev); } else {

        break; }

}
```

```
int mcRegGetInfoStruct(  
    HMCREG hReg, reginfo_t *reginf);  
  
reginf, rc = mc.mcRegGetInfoStruct(  
    number hReg)  
  
struct reginfo {  
    char regName[80]; char regDesc[80]; int regType; HMCDEV regDev;  
    void *regUserData; int regInput; };  
  
typedef struct reginfo reginfo_t;  
  
HMCREG hReg = 0;  
  
reginfo_t reginf;  
  
  
  
while (mcSignalGetNextHandle(hDev, hReg, &hReg) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="green">// Get the info on the register.</font>  
        mcRegGetInfoStruct(hReg, ®inf); } else {  
  
        break; }  
  
    }  
}
```

```
int mcRegGetNextHandle(  
    HMCDEV hDev, HMCREG startReg, HMCREG *hReg);  
  
hReg, rc = mc.mcRegGetNextHandle(  
    number hDev, number startReg)  
  
HMCSIG hReg = 0;  
  
HMCDEV hDev;  
  
mcDeviceGetHandle(0, 0,&hDev); while (<font  
color="blue">mcRegisterGetNextHandle(hDev, hReg, &hReg)</font> ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="green">// do something with hReg.</font> } else {  
  
        break; }  
  
}
```

```
int mcRegGetUserData(  
    HMCREG hReg, void **data);
```

N/A

```
struct myDataStruct {  
    int myInt; char myString[80]; ...  
};
```

```
HMCREG hReg;
```

```
int mInst=0;
```

```
myDataStruct *data = NULL; void *dataPtr; <font color="green">// Get the  
handle to holding register 1 on SIM0.</font> if (mcRegGetHandle(mInst,  
"SIM0/HoldingReg1", &hReg) == MERROR_NOERROR) {
```

```
    mcRegGetUserData(hReg, &dataPtr); data = (myDataStruct *) dataPtr; }
```

```
int mcRegGetValue(  
    HMCREG hReg, double *value);  
  
value, rc = mcRegGetValue(  
    number hReg)  
  
<font color="green">// Get the register value.</font> MINSTANCE mInst  
= 0; HMCREG hReg; int rc;  
  
double value; if (mcRegGetHandle(mInst, "core/global/Version", &hReg)  
== MERROR_NOERROR) {  
  
    rc = mcRegGetValue(hReg, &value); }
```

```
int mcRegGetValueLong(  
    HMCREG hReg, double *value);  
  
value, rc = mcRegGetValueLong(  
    number hReg)  
  
<font color="green">// Get the register value.</font> MINSTANCE mInst  
= 0; HMCREG hReg; int rc;  
  
long value; if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = mcRegGetValueLong(hReg, &value); }
```

```
int mcRegGetValueString(
    HMCREG hReg, char *buf, size_t bufSize);

buf, rc = mcRegGetValueString(
    number hReg)

<font color="green">// Get the register value.</font> MINSTANCE mInst
= 0; HMCREG hReg;

int rc;

char value[80]; if (mcRegGetHandle(mInst, "core/global/Version", &hReg)
== MERROR_NOERROR) {

    rc = mcRegGetValueString(hReg, &value, sizeof(value)); }
```

```
int mcRegGetValueStringClear(
    HMCREG hReg, char *buf, size_t bufSize);

buf, rc = mcRegGetValueString(
    number hReg)

<font color="green">// Get the register value and clear it.</font>
MINSTANCE mInst = 0; HMCREG hReg;

int rc;

char value[80]; if (mcRegGetHandle(mInst, "core/global/Version", &hReg)
== MERROR_NOERROR) {

    rc = mcRegGetValueStringClear(hReg, &value, sizeof(value)); }
```

mcRegRegister

C/C++ Syntax:

```
int mcRegRegister(
    HMCDEV hDev,
    const char *regName,
    const char *regDesc,
    int regType,
    HMCREG *hReg);
```

LUA Syntax:

N/A

Description: Adds a register to the core.

Parameters:

Parameter	Description
hDev	The handle of the register's parent device.
regName	A string buffer specifying the name of the register.
regDesc	A string buffer specifying the description of the register.
regType	REG_TYPE_NONE, REG_TYPE_INPUT, REG_TYPE_OUTPUT, REG_TYPE_HOLDING, REG_TYPE_COMMAND, REG_TYPE_ENCODER.
hReg	The address of a HMCREG that receives the register handle.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_DEVICE_NOT_FOUND	The hDev parameter was 0 or invalid.
MERROR_INVALID_ARG	regName , regDesc , or hReg is NULL.

Notes:

The **hDev** parameter **MUST** be a valid device handle. Otherwise, the function will crash.

Registers can contain data of any type, even string data. The maximum string length is 4096 bytes for registers other than **REG_TYPE_COMMAND** type registers and 1024 bytes for the **REG_TYPE_COMMAND** type registers.

REG_TYPE_NONE specifies a register that is not typed. This kind of register is basically just a data container that will not notify any process if the data it stores changes.

REG_TYPE_INPUT specifies a register that can be mapped to a Mach input signal. Mach and any GUI front end is notified if the data in this type of register is changed with a **MSG_REG_CHANGED** message. This type of register usually contains boolean data.

REG_TYPE_OUTPUT specifies a register that can be mapped to a Mach output signal. The plugin that owns this register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**.

REG_TYPE_HOLDING specifies a general purpose register that cannot be mapped to either an input or output signal. The plugin that owns this

register is notified with a **MSG_REG_CHANGED** synchronous message. All other plugins and the GUI are notified asynchronously with **MSG_REG_CHANGED**. This type of register usually contains boolean data.

REG_TYPE_COMMAND specifies a special purpose register that can be used to implement a simple command/response communication mechanism. Only the plugin that owns the register is notified with a **MSG_REG_COMMAND** synchronous message. It works with

- [mcRegSendCommand](#)
on the client side of the communication and
- [mcRegGetCommand](#)
and
- [mcRegSetResponse](#)
on the plugin side.

REG_TYPE_ENCODER specifies a register that can be mapped to a MPG or used to display the encoder counts in the GUI. Only Mach and the GUI are notified with a **MSG_REG_CHANGED** synchronous message.

Usage:

```
HMCREG m_hReg;  
HMCDEV m_hDev; // Contains the device handle from the device  
registration.  
int rc = mcRegRegister(m_hDev, "HoldingReg1", "HoldingReg1",  
REG_TYPE_HOLDING, &hReg);
```

```
int mcRegSendCommand(
    HMCREG hReg, const char *command, char *response, size_t
responseLen);

response, rc = mc.mcRegSendCommand(
    hReg, command)

<font color="green">// Send a command via a register.</font>
MINSTANCE mInst = 0;

if (mcRegGetHandle(mInst, "Sim0/SimCommand", &hReg) ==
MERROR_NOERROR) {

    char response[1024]; rc = <font
color="blue">mcRegSendCommand(hReg, "THC ON", response,
sizeof(response))</font>; if (rc == MERROR_NOERROR) {

        <font color="green">// check response.</font> }

    }
```

```
int mcRegSetDesc(  
    HMCREG hReg, const char *desc);  
  
rc = mc.mcRegSetDesc(  
    number hReg, string desc)  
  
<font color="green">// Set/change the description of the register.</font>  
MINSTANCE mInst = 0; if (mcRegGetHandle(mInst,  
"core/global/Version", &hReg) == MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetDesc(hReg, "Core Version")</font>; }  
}
```

```
int mcRegSetName(  
    HMCREG hReg, const char *name);  
  
rc = mc.mcRegSetName(  
    number hReg, string name)  
  
<font color="green">// Set/change the name of the register.</font>  
MINSTANCE mInst = 0; if (mcRegGetHandle(mInst,  
"core/global/Version", &hReg) == MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetName(hReg, "Version2")</font>;  
<font color="green">// The register path is now "core/global/Version2"  
</font> }
```

```
int mcRegSetResponse(
```

```
    HMCREG hReg,
```

```
    char *response);
```

N/A

```
<font color="green">// Set the response to a register command.</font> int
```

```
simControl::ProcessMsg(long msg, long param1, long param2) {
```

```
    HMCREG hReg = (HMCREG)param1; long RegVal;
```

```
    switch (msg) {
```

```
        case MSG_REG_COMMAND: mcRegGetValueLong(hReg, &RegVal);  
        if (hReg = m_RegCommand) { <font color="green">// Is this our command  
        register?</font> char command[1024]; mcRegGetCommand(hReg,  
        command, sizeof(command)); wxString cmd(command);  
        cmd.MakeUpper(); if (cmd == wxT("THC ON")) {
```

```
            m_thc = true; mcRegSetResponse(hReg, "OK"); } else if (cmd ==  
            wxT("THC OFF")) {
```

```
            m_thc = false; mcMotionSync(m_cid); mcRegSetResponse(hReg, "OK");  
            } else if (cmd == wxT("THC STATUS")) {
```

```
                if (m_thc) {
```

```
                    mcRegSetResponse(hReg, "1"); } else {
```

```
                    mcRegSetResponse(hReg, "0"); }
```

```
                }
```

```
            }
```

```
            break;
```

```
default:  
;  
}  
  
return(MERROR_NOERROR); }
```

```
int mcRegSetType(  
    HMCREG hReg, int regType);  
  
rc = mc.mcRegSetType(  
    number hReg, number regType)  
  
<font color="green">// Set the register type.</font> MINSTANCE mInst =  
0; if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetType(hReg, REG_TYPE_HOLDING)</font>; }  
}
```

```
int mcRegSetUserData(  
    HMCREG hReg, void *data);
```

N/A

```
struct myDataStruct {  
    int myInt; char myString[80]; ...  
};
```

```
HMCREG hReg;
```

```
MINSTANCE mInst = 0; myDataStruct data; <font color="green">// Get  
the handle to holding register 1 on SIM0.</font> mcRegGetHandle(mInst,  
"SIM0/HoldingReg1", &hReg); <font color="green">// Set the user data  
pointer to the address of data.</font> <font  
color="blue">mcRegSetUserData(hReg, &data)</font>;
```

```
int mcRegSetValue(  
    HMCREG hReg, double value);  
  
rc = mc.mcRegSetValue(  
    number hReg, number value)  
  
<font color="green">// Set the register value.</font> MINSTANCE mInst =  
0; if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetValue(hReg, 4.0)</font>; }  
  
}
```

```
int mcRegSetValueLong(  
    HMCREG hReg, double value);  
  
rc = mc.mcRegSetValueLong(  
    number hReg, number value)  
  
<font color="green">// Set the register value.</font> MINSTANCE mInst =  
0;  
  
if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetValueLong(hReg, 4)</font>; }  
}
```

```
int mcRegSetValueString(  
    HMCREG hReg, const char *value);  
  
rc = mc.mcRegSetValueString(  
    number hReg, string value)  
  
<font color="green">// Set the register value.</font> MINSTANCE mInst =  
0;  
  
if (mcRegGetHandle(mInst, "core/global/Version", &hReg) ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcRegSetValueString(hReg, "4.0")</font>; }  
}
```

```
int mcRegUnregister(  
    HMCDEV hDev, HMCREG hReg);
```

N/A

```
MINSTANCE mInst = 0; HMCREG hReg char *path = "Sim0/Register1";
```

```
mcRegGetHandle(mInst, path, &hReg); <font  
color="blue">mcRegUnregister(m_hDev, hReg);</font>
```

```
int mcScriptDebug(  
    MINSTANCE mInst, const char *filename);
```

N/A

```
<font color="green">// Debug the M6 macro</font> MINSTANCE mInst =  
0; int rc = mcScriptDebug(mInst, "m6.mcs");
```

```
int mcScriptEngineRegister(
```

```
    MINSTANCE mInst, HMCPLUG pluginid, const char *EngineName,  
    const char *EngineDesc, const char *FileExtensions);
```

N/A

```
<font color="green">// Register the LUA script engine.</font>  
MINSTANCE mInst = 0;
```

```
HMCPLUG id; <font color="green">// From mcPluginInit()</font>; int rc  
= mcScriptEngineRegister(mInst, id, "wxLua", "wxLua script engine",  
"mcs|lua|wx.lua");
```

```
int mcScriptExecute(  
    MINSTANCE mInst, const char *filename, BOOL async);  
  
rc = mc.mcScriptExecute(  
    number mInst, string filename, number async)  
  
<font color="green">// Execute the M6 script and wait on it to complete.  
</font> MINSTANCE mInst = 0;  
  
rc = mcScriptExecute(mInst, "m6.mcs", FALSE);
```

```
int mcScriptExecuteIfExists(  
    MINSTANCE mInst, const char *filename, BOOL async);  
  
rc = mc.mcScriptExecuteIfExists(  
    number mInst, string filename, number async)  
  
<font color="green">// Execute the M6 script and wait on it to complete.  
</font> MINSTANCE mInst = 0;  
  
rc = mcScriptExecute(mInst, "m6.mcs", FALSE);
```

```
int mcScriptExecutePrivate(  
    MINSTANCE mInst, const char *filename, BOOL async);  
  
rc = mc.mcScriptExecutePrivate(  
    number mInst, string filename, BOOL async)  
  
<font color="green">// Execute myScript.mcs asynchronously in it's own  
private environment.</font> MINSTANCE mInst = 0;  
  
int rc = mcScriptExecutePrivate(mInst, "myScript.mcs", TRUE);
```

```
int mcScriptGetExtensions(MINSTANCE mInst, char *buf, long bufsize);
```

N/A

```
<font color="green">// Retrieve all of the registered script extensions.  
</font> MINSTANCE mInst = 0; mcScriptGetExtensions(MINSTANCE  
mInst, char *buf, long bufsize);
```

```

int mcSignalEnable(
    HMCSIG hSig, BOOL enabled);

rc = mc.mcSignalEnable(
    number hSig, number enabled)

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {

    m_cid = mInst; m_id = id;

    m_timer = new simTimer(this); m_cycletime = .001; HMCSIG hSig;

    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",
        DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);

    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,
        &m_Input0); mcIoRegister(m_hDev, "Output0", "Output0",
        IO_TYPE_OUTPUT, &m_Output0)

    if (mcSignalGetHandle(m_cid, ISIG_INPUT1, int sigid, &hSig) ==
        MERROR_NOERROR) {

        if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {

            <font color="green">// Signal mapped successfully</font> <font
            color="blue">mcSignalEnable(hSig, TRUE);</font> <font
            color="green">// Enable the signal.</font> }

        }
    }
}

```

```

int mcSignalGetHandle(
    MINSTANCE mInst, int sigid,
    HMCSIG *hSig);

hSig, rc = mc.mcSignalGetHandle(
    number mInst, number sigid)

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {
    m_cid = mInst; m_id = id;
    m_timer = new simTimer(this); m_cycletime = .001; HMCSIG hSig;
    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",
        DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);
    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,
        &m_Input0); mcIoRegister(m_hDev, "Output0", "Output0",
        IO_TYPE_OUTPUT, &m_Output0)
    if (<font color="blue">mcSignalGetHandle(m_cid, ISIG_INPUT1,
        &hSig)</font> == MERROR_NOERROR) {
        if (mcSignalMap(hSig, m_Input0) == MERROR_NOERROR) {
            <font color="green">// Signal mapped successfully.</font>
            mcSignalEnable(hSig, true); <font color="green">// Enable the signal.
            </font> }
    }
}

```

```
int mcSignalGetInfo(  
    HMCSIG hSig, int *enabled, char *name, size_t namelen, char *desc,  
    size_t descLen, int *activeLow);  
  
enabled, name, desc, activeLow, rc = mc.mcSignalGetInfo(number hSig)  
  
HMCSIG hSig = 0;  
  
int enabled = 0;  
  
int nameLen = 20  
  
char nameBuf[nameLen]; int descLen = 40;  
  
char descBuf[descLen]; int activeLow = 0;  
  
  
  
while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcSignalGetInfo(hSig, &enabled, nameBuf,  
        nameLen, descBuf, descLen, &activeLow)</font>; } else {  
  
        break; }  
  
    }  
}
```

```
int mcSignalGetInfoStruct(  
    HMCSIG hSig, siginfo_t *siginf);  
  
siginf, rc = mc.mcSignalGetInfoStruct(  
    number hSig)  
  
struct siginfo {  
    char sigName[80]; char sigDesc[80]; int sigEnabled; int sigActiveLow;  
    HMCI0 sigMappedIoHandle; };  
  
typedef struct siginfo siginfo_t;  
  
HMCSIG hSig = 0;  
  
siginfo_t siginf;  
  
  
  
while (mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcSignalGetInfoStruct(hSig, &siginf);</font> } else  
{  
  
    break; }  
  
}  
}
```

```
int mcSignalGetNextHandle(  
    MINSTANCE mInst, int sigtype, HMCSIG startSig, HMCSIG *hSig);  
  
hSig, rc = mc.mcSignalGetNextHandle(  
    number mInst, number sigtype, number startSig)  
  
HMCSIG hSig = 0;  
  
  
  
while (<font color="blue">mcSignalGetNextHandle(0,  
    SIG_TYPE_INPUT, hSig, &hSig)</font> == MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="green">// Do something with hSig.</font> } else {  
  
        break; }  
  
    }  
}
```

```
int mcSignalGetState(  
    HMCSIG hSig, BOOL *state)  
  
state, rc = mc.mcSignalGetState(  
    number hSig)  
  
HMCSIG hSig = 0;  
  
BOOL sigState = FALSE;  
  
<font color="green">// Get the state of all input signals.</font> while  
(mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcSignalGetState(hSig, &sigState);</font> } else {  
  
        break; }  
  
}
```

```
int mcSignalHandleWait(  
    HMCSIG hSig, int waitMode, double timeoutSecs);  
  
rc = mc.mcSignalHandleWait(  
    number hSig, number waitMode, number timeoutSecs);  
  
MINSTANCE mInst = 0;  
  
HMCSIG hSig;  
  
int rc;  
  
rc = mcSignalGetHandle(mInst, ISIG_INPUT1 &hSig); if (rc ==  
MERROR_NOERROR) {  
  
    rc = <font color="blue">mcSignalHandleWait(hSig,  
WAIT_MODE_HIGH, .5)</font>; switch (rc) {  
  
        case MERROR_NOERROR: <font color="green">// Signal changed  
state from low to high</font> break; case MERROR_TIMED_OUT: <font  
color="green">// The signal didn't change state in the allotted time.</font>  
break; case MERROR_NOT_ENABLED: <font color="green">// The  
control was not enabled at the time of the function call or the control was  
disabled during the function call.</font> break; }  
  
}
```

```
int mcSignalMap(
    HMCSIG hSig, HMCIO hIo);
rc = mc.mcSignalMap(
    number hSig, number hIo)

simControl::simControl(MINSTANCE mInst, HMCPLUG id) {
    m_cid = mInst; m_id = id; m_timer = new simTimer(this); m_cycletime
    = .001; HMCSIG hSig;

    mcDeviceRegister(m_cid, m_id, "Sim0", "Simulation Device",
    DEV_TYPE_MOTION | DEV_TYPE_IO, &m_hDev);

    mcIoRegister(m_hDev, "Input0", "Input0", IO_TYPE_INPUT,
    &m_Input0); mcIoRegister(m_hDev, "Output0", "Output0",
    IO_TYPE_OUTPUT, &m_Output0)

    if (mcSignalGetHandle(m_cid, ISIG_INPUT1, &hSig) ==
    MERROR_NOERROR) {

        if (<font color="blue">mcSignalMap(hSig, m_Input0)</font> ==
        MERROR_NOERROR) {

            <font color="green">// Signal mapped successfully</font> }

    }
}
```

```
int mcSignalSetActiveLow(
    HMCSIG hSig, BOOL activelow);
rc = mc.mcSignalSetActiveLow(
    number hSig, number activelow)
HMCSIG hSig = 0;
BOOL activeLow = TRUE;

<font color="green">// Set all input signals active low.</font> while
(mcSignalGetNextHandle(0, SIG_TYPE_INPUT, hSig, &hSig) ==
MERROR_NOERROR) {

    if (hSig != 0) {

        <font color="blue">mcSignalSetActiveLow(hSig, activeLow);</font> }
    else {

        break; }

}
```

```
int mcSignalSetState(  
    HMCSIG hSig, BOOL enabled);  
  
rc = mc.mcSignalSetState(  
    number hSig, number enabled)  
HMCSIG hSig = 0;  
  
<font color="green">// Set all output signals inactive.</font> while  
(mcSignalGetNextHandle(0, SIG_TYPE_OUTPUT, hSig, &hSig) ==  
MERROR_NOERROR) {  
  
    if (hSig != 0) {  
  
        <font color="blue">mcSignalSetState(hSig, FALSE);</font> } else {  
  
        break; }  
  
}
```

```
int mcSignalUnmap(
    HMCSIG hSig);

rc = mc.mcSignalUnmap(
    number hSig)

MINSTANCE mInst = 0; HMCSIG hSig;

if (mcSignalGetHandle(mInst, ISIG_INPUT1, &hSig) ==
MERROR_NOERROR) {

    if (<font color="blue">mcSignalUnmap(hSig)</font> ==
MERROR_NOERROR) {

        <font color="green">// Signal now has no I/O mappings</font> }

    }
}
```

```
int mcSignalWait(  
    MINSTANCE mInst, int sigId, int waitMode, double timeoutSecs);  
  
rc = mc.mcSignalWait(  
    number mInst, number sigId, number waitMode, number timeoutSecs);  
  
MINSTANCE mInst = 0; HMCSIG hSig;  
  
int rc;  
  
  
  
rc = <font color="blue">mcSignalWait(mInst, ISIG_INPUT1,  
    WAIT_MODE_HIGH, .5)</font>; switch (rc) {  
  
    case MERROR_NOERROR: <font color="green">// Signal changed state  
        from low to high</font> break; case MERROR_TIMED_OUT: <font  
        color="green">// The signal didn't change state in the allotted time.</font>  
        break; case MERROR_NOT_ENABLED: <font color="green">// The  
        control was not enabled at the time of the function call or the control was  
        disabled during the function call.</font> break; }
```

```
int mcSoftLimitGetState(  
    MINSTANCE mInst, int axis, double *ison);  
  
ison, rc = mc.mcSoftLimitGetState(  
    number mInst, number axis)  
  
int mInst = 0;  
  
int axis = Z_AXIS;  
  
double IsOn = 0; <font color="green">// DO_OFF to turn off.</font>  
mcSoftLimitSetState(mInst, axis, &IsOn); <font color="green">// Get the  
softlimit for the Z axis return will be 1 or 0 1==on.</font>
```

```
int mcSoftLimitMaxMinsClear(  
    MINSTANCE mInst);  
  
rc = mc.mcSoftLimitMaxMinsClear(  
    number mInst)  
  
int mInst=0;  
  
<font color="green">// Clear the Softlimits for the file so they will be  
recalculated when a regen is done.</font>  
mcSoftLimitMaxMinsClear(mInst);
```

```
int mcSoftLimitSetState(  
    MINSTANCE mInst, int axis, int on);  
  
rc = mc.mcSoftLimitSetState(  
    number mInst, number axis, number on)  
  
int mInst=0;  
  
int axis = Z_AXIS;  
  
int TurnOn = MC_ON; <font color="green">// MC_OFF to turn off.</font>  
mcSoftLimitSetState(mInst, axis, TurnOn); <font color="green">// Set the  
softlimit forthe Z axis to be ON.</font>
```

```
int mcSpindleCalcCSSToRPM(  
    MINSTANCE mInst, double DiaOfCut, BOOL Inch);  
  
rc = mc.mcSpindleCalcCSSToRPM(  
    number mInst, number DiaOfCut, number Inch);  
  
<font color="green">// make the spindle speed calc the correct RPM for a  
.550" inch cut.</font> MINSTANCE mInst = 0;  
  
int rc = mcSpindleCalcCSSToRPM(mInst, .550, TRUE);
```

```
int mcSpindleGetAccelTime(  
    MINSTANCE mInst, int Range, double *Sec);  
  
Sec, rc = mc.mcSpindleGetAccelTime(  
    number mInst, number Range)  
  
<font color="green">// Get the accel time for spindle range 2.</font>  
MINSTANCE mInst = 0;  
  
double sec = 0;  
  
int rc = mcSpindleGetAccelTime(mInst, 2, &sec);
```

```
int mcSpindleGetCommandRPM(  
    MINSTANCE mInst, double *RPM);  
  
RPM, rc = mc.mcSpindleGetCommandRPM(  
    number mInst)  
  
<font color="green">// Get the current commanded RPM</font>  
MINSTANCE mInst = 0; double rpm = 0.0;  
  
int rc = mcSpindleGetCommandRPM(mInst, &rpm);
```

```
int mcSpindleGetCurrentRange(  
    MINSTANCE mInst, int *Range);  
  
Range, rc = mc.mcSpindleGetCurrentRange(  
    number mInst)  
  
<font color="green">// Get the current spindle range.</font> MINSTANCE  
mInst = 0;  
  
int range = 0;  
  
int rc = mcSpindleGetCurrentRange(mInst, &range);
```

```
int mcSpindleGetDecelTime(  
    MINSTANCE mInst, int Range, double *Sec);  
  
Sec, rc = mc.mcSpindleGetDecelTime(  
    number mInst, number Range)  
  
<font color="green">// Get the decel. time for spindle range 3</font>  
MINSTANCE mInst = 0;  
  
double sec = 0.0;  
  
int rc = mcSpindleGetDecelTime(mInst, 3, &ec);
```

```
int mcSpindleGetDirection(  
    MINSTANCE mInst, int *dir);  
  
dir, rc = mc.mcSpindleGetDirection(  
    number mInst)  
  
<font color="green">// Get the current spindle direction.</font>  
MINSTANCE mInst = 0;  
  
int dir = MC_SPINDLE_OFF; int rc = mcSpindleGetDirection(mInst,  
&dir);
```

```
int mcSpindleGetFeedbackRatio(  
    MINSTANCE mInst, int Range, double *Ratio);  
  
Ratio, rc = mc.mcSpindleGetFeedbackRatio(  
    number mInst, number Range)  
  
<font color="green">// Get the feedback ratio for spindle range 0.</font>  
MINSTANCE mInst = 0;  
  
double ratio = 0.0;  
  
int rc = mcSpindleGetFeedbackRatio(mInst, 0, &ratio);
```

```
int mcSpindleGetMaxRPM(  
    MINSTANCE mInst, int Range, double *MaxRPM);  
  
MaxRPM, rc = mc.mcSpindleGetMaxRPM(  
    number mInst, number Range)  
  
<font color="green">// Get the max RPM for range 0.</font>  
MINSTANCE mInst = 0;  
  
double mrpm = 0.0  
  
int rc = mcSpindleGetMaxRPM(mInst, 0, &mrpm);
```

```
int mcSpindleGetMinRPM(  
    MINSTANCE mInst, int Range, double *MinRPM);  
  
MinRPM, rc = mc.mcSpindleGetMinRPM(  
    number mInst, number Range)  
  
<font color="green">// Get the min RPM for range 0.</font> MINSTANCE  
mInst = 0;  
  
double mrpm = 0.0  
  
int rc = mcSpindleGetMinRPM(mInst, 0, &mrpm);
```

```
int mcSpindleGetMotorAccel(  
    MINSTANCE mInst, int Range, double *Accel);  
  
Accel, rc = mc.mcSpindleGetMotorAccel(  
    number mInst, number Range)  
  
<font color="green">// Get the spindle acceleration for range 1.</font>  
MINSTANCE mInst = 0;  
  
double accel = 0.0;  
  
int rc = mcSpindleGetMotorAccel(mInst, 1, &accel);
```

```
int mcSpindleGetMotorMaxRPM(  
    MINSTANCE mInst, double *RPM);  
  
RPM, rc = mc.mcSpindleGetMotorMaxRPM(  
    number mInst)  
  
<font color="green">// Get the maximum spindle motor RPM.</font>  
MINSTANCE mInst = 0;  
  
double rpm = 0.0;  
  
int rc = mcSpindleGetMotorMaxRPM(mInst, &rpm);
```

```
int mcSpindleGetMotorRPM(  
    MINSTANCE mInst, double *RPM);  
  
rpm, rc = mc.mcSpindleGetMotorRPM(  
    number mInst)  
  
<font color="green">// Get the current motor RPM.</font> MINSTANCE  
mInst = 0;  
  
double rpm = 0.0;  
  
int rc = mcSpindleGetMotorRPM(mInst, &rpm);
```

```
int mcSpindleGetOverride(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mcSpindleGetOverride(  
    number mInst)  
  
<font color="green">// Get the spindle override. .5 == 50%</font>  
MINSTANCE mInst = 0;  
  
double percent = 0.0;  
  
int rc = mcSpindleGetOverride(mInst, &percent);
```

```
int mcSpindleGetOverrideEnable(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mcSpindleGetOverrideEnable(  
    number mInst)  
  
<font color="green">// Get the spindle override status.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcSpindleGetOverrideEnable(mInst, &enabled);
```

```
int mcSpindleGetReverse(  
    MINSTANCE mInst, int Range, BOOL *Reversed);  
  
BOOL Reversed, rc = mc.mcSpindleGetReverse(  
    number mInst, number Range)  
  
<font color="green">// See if the spindle is reversed in range 1</font>  
MINSTANCE mInst = 0;  
  
BOOL reversed = FALSE; int mcSpindleGetReverse(mInst, 1, &reversed);
```

```
int mcSpindleGetSensorRPM(  
    MINSTANCE mInst, double *RPM);  
  
RPM, rc = mc.mcSpindleGetSensorRPM(  
    number mInst)  
  
<font color="green">// Get the spindle sensor RPM.</font> MINSTANCE  
mInst = 0;  
  
double rpm = 0.0;  
  
int rc = mcSpindleGetSensorRPM(mInst, &rpm);
```

```
int mcSpindleGetSpeedCheckEnable(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mc.mcSpindleGetSpeedCheckEnable(  
    number mInst)  
  
<font color="green">// Check the staus of spindle speed check.</font>  
MINSTANCE mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcSpindleGetSpeedCheckEnable(mInst, &enabled);
```

```
int mcSpindleGetSpeedCheckPercent(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mc.mcSpindleGetSpeedCheckPercent(  
    number mInst)  
  
<font color="green">// Get the speed check percentage.</font>  
MINSTANCE mInst = 0;  
  
double percent = 0.0;  
  
int rc = mcSpindleGetSpeedCheckPercent(mInst, &percent);
```

```
int mcSpindleGetTrueRPM(  
    MINSTANCE mInst, double *RPM);  
  
RPM, rc = mc.mcSpindleGetTrueRPM(  
    number mInst)  
  
<font color="green">// Get the true spindle RPM.</font> MINSTANCE  
mInst = 0;  
  
double rpm = 0.0;  
  
int rc = mcSpindleGetTrueRPM(mInst, &rpm);
```

```
int mcSpindleSetAccelTime(  
    MINSTANCE mInst, int Range, double Sec);  
  
rc = mc.mcSpindleSetAccelTime(  
    number mInst, number Range, number Sec);  
  
<font color="green">// Set the accel. time for spindle range 0.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetAccelTime(mInst, 0, 5.0);
```

```
int mcSpindleSetCommandRPM(  
    MINSTANCE mInst, double RPM);  
  
rc = mc.mcSpindleSetCommandRPM(  
    number mInst, number RPM)  
  
<font color="green">// Set the spindle RPM to 5000.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcSpindleSetCommandRPM(mInst, 5000);
```

```
int mcSpindleSetDecelTime(  
    MINSTANCE mInst, int Range, double Sec);  
  
rc = mc.mcSpindleSetDecelTime(  
    number mInst, number Range, number Sec)  
  
<font color="green">// Set the decel. time for spindle range 0.</font>  
MINSTANCE mInst = 0;  
  
in rc = mcSpindleSetDecelTime(mInst, 0, 5);
```

```
int mcSpindleSetDirection(  
    MINSTANCE mInst, int dir);  
  
rc = mc.mcSpindleSetDirection(  
    number mInst, number dir)  
  
<font color="green">// Set the spindle direction to FWD</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetDirection(mInst, MC_SPINDLE_FWD);
```

```
int mcSpindleSetDirectionWait(  
    MINSTANCE mInst, int dir);  
  
rc = mc.mcSpindleSetDirectionWait(  
    number mInst, number dir)  
  
<font color="green">// Set the spindle direction to FWD and wait</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetDirectionWait(mInst, MC_SPINDLE_FWD);
```

```
int mcSpindleSetFeedbackRatio(  
    MINSTANCE mInst, int Range, double Ratio);  
  
rc = mc.mcSpindleSetFeedbackRatio(  
    number mInst, number Range, number Ratio)  
  
<font color="green">// Set the feedback ratio for spindle range 0.</font>  
MINSTANCE mInst = 0;  
  
int mcSpindleSetFeedbackRatio(mInst, 0, 1.2); <font color="green">// 1.2 :  
1</font>
```

```
int mcSpindleSetMaxRPM(  
    MINSTANCE mInst, int Range, double RPM);  
  
rc = mc.mcSpindleSetMaxRPM(  
    number mInst, number Range, number RPM);  
  
<font color="green">// Set the max RPM for spindle range 0 to  
5000</font> MINSTANCE mInst = 0;  
  
int mcSpindleSetMaxRPM(mInst, 0, 5000);
```

```
int mcSpindleSetMinRPM(  
    MINSTANCE mInst, int Range, double RPM);  
  
rc = mc.mcSpindleSetMinRPM(  
    number mInst, number Range, number RPM);  
  
<font color="green">// Set the min RPM for spindle range 0 to 60</font>  
MINSTANCE mInst = 0;  
  
int mcSpindleSetMinRPM(mInst, 0, 60);
```

```
int mcSpindleSetMotorAccel(  
    MINSTANCE mInst, int Range, double Accel);  
  
rc = mc.mcSpindleSetMotorAccel(  
    number mInst, number Range, number Accel);  
  
<font color="green">// Set the acceleration for the spindle motor for range  
0</font> MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetMotorAccel(mInst, 0, 10000);
```

```
int mcSpindleSetMotorMaxRPM(  
    MINSTANCE mInst, double RPM);  
  
rc = mc.mcSpindleSetMotorMaxRPM(  
    number mInst, number RPM)  
  
<font color="green">// Cap the spindle motor RPM to 6000.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetMotorMaxRPM(mInst, 6000);
```

```
int mcSpindleSetOverride(  
    MINSTANCE mInst, double percent);  
  
rc = mc.mcSpindleSetOverride(  
    number mInst, number percent)  
  
<font color="green">// Set the spindle override to 50% (.5 == 50%)</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetOverride(mInst, 0.5);
```

```
int mcSpindleSetOverrideEnable(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcSpindleSetOverrideEnable(  
    number mInst, number enable)  
  
<font color="green">// Enable spindle override.</font> MINSTANCE  
mInst = 0;  
  
int rc = mcSpindleSetOverrideEnable(mInst, TRUE);
```

```
int mcSpindleSetRange(  
    MINSTANCE mInst, int Range);  
  
rc = mc.mcSpindleSetRange(  
    number mInst, number Range)  
  
<font color="green">// Set the current spindle range to the first range (0).  
</font> MINSTANCE mInst = 0;  
  
mcSpindleSetRange(mInst, 0);
```

```
int mcSpindleSetReverse(  
    MINSTANCE mInst, int Range, BOOL Reversed);  
  
rc = mc.mcSpindleSetReverse(  
    number mInst, number Range, number Reversed)  
  
<font color="green">// Set spindle range 1 as being reversed.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetReverse(mInst, 1, TRUE);
```

```
int mcSpindleSetSensorRPM(  
    MINSTANCE mInst, double RPM);  
  
rc = mc.mcSpindleSetSensorRPM(  
    number mInst, number RPM)  
  
<font color="green">// Set the spindle sensor RPM to 1000</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetSensorRPM(mInst, 1000);
```

```
int mcSpindleSetSpeedCheckEnable(  
    MINSTANCE mInst, BOOL enable);  
  
rc = mc.mcSpindleSetSpeedCheckEnable(  
    number mInst, number enable)  
  
<font color="green">// Enable spindle speed checking.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetSpeedCheckEnable(mInst, TRUE);
```

```
int mcSpindleSetSpeedCheckPercent(  
    MINSTANCE mInst, double percent);  
  
rc = mc.mcSpindleSetSpeedCheckPercent(  
    number mInst, number percent)  
  
<font color="green">// Set the speed check percentage to 5%. 100 == 100%  
</font> MINSTANCE mInst = 0;  
  
int rc = mcSpindleSetSpeedCheckPercent(mInst, 5);
```

```
int mcSpindleSpeedCheck(  
    MINSTANCE mInst, int *SpeedOk);  
  
SpeedOk, rc = mc.mcSpindleSpeedCheck(  
    number mInst)  
  
<font color="green">// Check to see if the spidnle speed is in the allowed  
range.</font> MINSTANCE mInst = 0; BOOL speedOk;  
  
int rc = mcSpindleSpeedCheck(mInst, &speedOk);
```

```
int mcToolGetCurrent(  
    MINSTANCE mInst, int *toolnum);  
  
toolnum, rc = mc.mcToolGetCurrent(  
    number mInst)  
  
<font color="green">// Get the current tool.</font> MINSTANCE mInst =  
0; int toolnum = 0;  
  
int rc = mcToolGetCurrent(mInst, &toolnum);
```

```
int mcToolGetData(  
    MINSTANCE mInst, int type, int toolNumber, double *value);  
  
value, rc = mc.mcToolGetData(  
    number mInst, number type, number toolNumber)  
  
MINSTANCE mInst = 0; int toolnum = 5;  
  
double val = 0;  
  
<font color="green">// Get the tool height offset for tool number 5.</font>  
int rc = mcToolGetData(mInst, MTOOL_MILL_HEIGHT, toolnum, &val);
```

```
int mcToolGetDataExDbl(
    MINSTANCE mInst, int toolnum, const char *fieldName, double
    *value);

value, rc = mc.mcToolGetDataExDbl(
    number mInst, number toolNumber, string fieldName)

MINSTANCE mInst = 0;

int toolnum = 5;

double val = 0;

<font color="green">// Get the contents of "MyUserField" for tool number
5.</font> int rc = mcToolGetDataExDbl(mInst, toolnum, "MyUserField",
&val);
```

```
int mcToolGetDataExInt(  
    MINSTANCE mInst, int toolnum, const char *fieldName, int *value);  
  
value, rc = mc.mcToolGetDataExInt(  
    number mInst, number toolNumber, string fieldName)  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
int val = 0;  
  
<font color="green">// Get the contents of "MyUserField" for tool number  
5.</font> int rc = mcToolGetDataExInt(mInst, toolnum, "MyUserField",  
&val);
```

```
int mcToolGetDataExStr(  
    MINSTANCE mInst, int toolnum, const char *fieldName, char *value,  
    size_t valueLen);  
  
value, rc = mc.mcToolGetDataExStr(  
    number mInst, number toolNumber, string fieldName)  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
char val[128];  
  
<font color="green">// Get the contents of "MyUserField" for tool number  
5.</font> int rc = mcToolGetDataExStr(mInst, toolnum, "MyUserField",  
val, sizeof(val));
```

```
int mcToolGetDesc(  
    MINSTANCE mInst, int toolnum, char *buff, size_t bufsize);  
  
buff, rc = mc.mcToolGetDesc(  
    number mInst, number toolnum)  
  
MINSTANCE mInst = 0; char desc[128];  
  
int toolnum = 5;  
  
<font color="green">// Get the description of tool number 5 for controller  
0.</font> int rc = mcToolGetDesc(mInst, toolnum, desc, sizeof(desc));
```

```
int mcToolGetSelected(  
    MINSTANCE mInst, int *toolnum);  
  
toolnum, rc = mc.mcToolGetSelected(  
    number mInst)  
  
<font color="green">// Get the selected tool number.</font> MINSTANCE  
mInst = 0;  
  
int toolnum = 0;  
  
int rc = mcToolGetSelected(mInst, &toolnum);
```

```
int mcToolLoadFile(  
    MINSTANCE mInst, const char *FileToLoad);  
  
rc = mc.mcToolLoadFile(  
    number mInst, string FileToLoad)  
  
<font color="green">// Load a tool table.</font> MINSTANCE mInst = 0;  
  
int rc = mcToolLoadFile(mInst, "tooltable.xls");
```

```
int mcToolPathCreate(  
    MINSTANCE mInst, void *window);
```

N/A

```
wxPanel* m_tpTP;
```

```
MINSTANCE mInst = 0; m_tpTP = new wxPanel(itemPanel129,  
ID_TOOLPATH_PANEL, wxDefaultPosition, wxDefaultSize,  
wxTAB_TRAVERSAL ); rc = m_tpTP->Show(); <font color="green">//  
Pass the handle of the tool path.</font> mcToolPathCreate(mInst, m_tpTP-  
>GetHWND());
```

```
int mcToolPathDelete(  
    MINSTANCE mInst, void *parent);
```

N/A

```
<font color="green">// Delete an existing tool path.</font> MINSTANCE  
mInst = 0; HWND parent; <font color="green">// A previously valid  
window handle.</font> int rc = mcToolPathDelete(mInst, parent);
```

```
int mcToolPathGenerate(  
    MINSTANCE mInst);  
  
rc = mc.mcToolPathGenerate(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
<font color="green">// Regenerate the toolpaths for controller instance  
0</font> int rc = mcToolPathGenerate(mInst);
```

```
int mcToolPathGenerateAbort(  
    MINSTANCE mInst);  
  
rc = mc.mcToolPathGenerateAbort(  
    number mInst);  
  
MINSTANCE mInst = 0;  
  
<font color="green">// Abort the regeneration of the toolpath for controller  
0.</font> int rc = mcToolPathGenerateAbort(mInst);
```

```
int mcToolPathGeneratedPercent(  
    MINSTANCE mInst, double *percent);  
  
percent, rc = mc.mcToolPathGeneratedPercent(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
double pdone = 0;  
  
  
  
while (pdone != 100) {  
  
    <font color="green">// Get the percent of the file that is loaded.</font>  
    mcToolPathGeneratedPercent(mInst, &pdone); <font color="green">// Post  
    the pdone to some dialog.</font> Sleep(250); }
```

```
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double  
*yPos, double *zPos);
```

```
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double  
*yPos, double *zPos);
```

```
<font color="green">// </font> MINSTANCE mInst = 0;  
mcToolPathGetAAxisPosition(MINSTANCE mInst, double *xPos, double  
*yPos, double *zPos);
```

mcToolPathGetARotationAxis

C/C++ Syntax:

```
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

LUA Syntax:

```
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

Usage:

```
//  
MINSTANCE mInst = 0;  
mcToolPathGetARotationAxis(MINSTANCE mInst, int *axis);
```

```
int mcToolPathGetAxisColor(
```

```
    MINSTANCE mInst, unsigned long *axiscolor, unsigned long
    *limitcolor);
```

```
axiscolor, limitcolor, rc = mc.mcToolPathGetAxisColor(
```

```
    number mInst)
```

```
<font color="green">// Get the axis and limit colors.</font> MINSTANCE
mInst = 0;
```

```
int rc = mcToolPathGetAxisColor(MINSTANCE mInst, unsigned long
    *axiscolor, unsigned long *limitcolor);
```

```
int mcToolPathGetBackColor(
    MINSTANCE mInst, unsigned long *topcolor, unsigned long *botcolor);
topcolor, botcolor, rc = mc.mcToolPathGetBackColor(
    number mInst)
MINSTANCE mInst = 0;
unsigned long topcol, botcol;

<font color="green">// Get the tool background top and bottom
colors</font> int rc = mcToolPathGetBackColor(m_inst, &topcol,
&botcol);
```

```
int mcToolPathGetDrawLimits(  
    MINSTANCE mInst, BOOL *drawlimits);  
  
drawlimits, rc = mcToolPathGetDrawLimits(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
BOOL drawlimits = FALSE;  
  
<font color="green">// See if the soft limits bounding box is beeing shown  
in the toolpath.</font> int rc = mcToolPathGetDrawLimits(mInst,  
&drawlimits);
```

```
int mcToolPathGetExecution(
```

```
    MINSTANCE mInst, unsigned long exNum, void **data, unsigned long
    *len);
```

N/A

```
struct Executions

{
    bool linear : 1; //Is this a linear move?

    bool rapid : 1; //Is it a Rapid move?

    bool inc : 1; //Is this an inc move?

    bool comp : 1; bool rotation : 1; //arc direction
    bool UsedAxis0 : 1; bool UsedAxis1 : 1;
    bool UsedAxis2 : 1; bool UsedAxis3 : 1; bool UsedAxis4 : 1;
    bool UsedAxis5 : 1; bool UsedAxis6 : 1; bool UsedAxis7 : 1;
    bool UsedAxis8 : 1; bool UsedAxis9 : 1; bool UsedAxis10 : 1;
    float end[6]; float center[3];
    float normal[3]; unsigned int LineNumber; // Line number in the
    file...

    unsigned int ExecutionID; // Number of the move from the Gcode
    interperter float FeedRate; // Feedrate of the move
    unsigned char ToolNumber; // Tool number of the move used to show offsets..

    unsigned char Fixture; // The fixture.

};

<font color="green">// </font> MINSTANCE mInst = 0;

unsigned long exNum = 0;

unsigned long len;

void *data;

int rc;

struct Executions *ex;
```

```
<font color="green">// Get the length of the move execution structure first.  
</font> while (mcToolPathGetExecution(mInst, exNum, NULL, &len) ==  
MERROR_NOERROR) {  
  
    <font color="green">// Allocate some mem to receive the data.</font>  
    data = malloc(len); if (data != NULL) {  
  
        <font color="green">// Get the data.</font> if  
(mcToolPathGetExecution(mInst, exNum, &data, &len) ==  
MERROR_NOERROR) {  
  
        <font color="green">// cast the void pointer to the current structure.  
</font> ex = (struct Executions *)data; ...  
  
        free(data); }  
  
    }  
}
```

```
int mcToolPathGetFollowMode(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mc.mcToolPathGetFollowMode(  
    number mInst)  
  
<font color="green">// See if jog follow is enabled.</font> MINSTANCE  
mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcToolPathGetFollowMode(mInst, &enabled);
```

```
int mcToolPathGetGenerating(  
    MINSTANCE mInst, int *pathGenerating);  
  
pathGenerating, rc = mc.mcToolPathGetGenerating(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
BOOL IsGen = FALSE;  
  
<font color="green">// See if the tool path is generating.</font> int rc =  
mcGetPathGenerating(mInst, &IsGen);
```

```
int mcToolPathGetLeftMouseDn(  
    MINSTANCE mInst, double *x, double *y, double *z);  
  
x, y, z, rc = mc.mcToolPathGetLeftMouseDn(  
    number mInst)  
  
<font color="green">// Get the mouse coordinates in the tool path.</font>  
MINSTANCE mInst = 0;  
  
double x, y, z;  
  
int rc = mcToolPathGetLeftMouseDn(MINSTANCE mInst, double *x,  
    double *y, double *z);
```

```
int mcToolPathGetLeftMouseUp(  
    MINSTANCE mInst, double *x, double *y, double *z);  
  
x, y, z, rc = mc.mcToolPathGetLeftMouseUp(  
    number mInst)  
  
<font color="green">// Get the mouse coordinates in the tool path.</font>  
MINSTANCE mInst = 0;  
  
double x, y, z;  
  
int rc = mcToolPathGetLeftMouseUp(MINSTANCE mInst, double *x,  
    double *y, double *z);
```

```
int mcToolPathGetPathColor(  
    MINSTANCE mInst, unsigned long *rapidcolor, unsigned long  
    *linecolor, unsigned long *arccolor, unsigned long *highlightcolor);  
  
rapidcolor, linecolor, arccolor, highlightcolor, rc =  
mcToolPathGetPathColor(  
    number mInst)  
  
MINSTANCE mInst = 0;  
  
unsigned long rapcol, lincol, arccol, highcol; <font color="green">// Get the  
tool path colors</font> int rc = mcToolPathGetPathColor(m_inst, &rapcol,  
&lincol, &arccol, &highcol);
```

```
int mcToolPathIsSignalMouseClicks(  
    MINSTANCE mInst, BOOL *enabled);  
  
enabled, rc = mcToolPathIsSignalMouseClicks(  
    number mInst)  
  
<font color="green">// See if we are signaling mouse click events in the  
tool path.</font> MINSTANCE mInst = 0;  
  
BOOL enabled = FALSE;  
  
int rc = mcToolPathIsSignalMouseClicks(mInst, &enabled);
```

```
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double  
yPos, double zPos);
```

```
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double  
yPos, double zPos);
```

```
<font color="green">// </font> MINSTANCE mInst = 0;  
mcToolPathSetAAxisPosition(MINSTANCE mInst, double xPos, double  
yPos, double zPos);
```

mcToolPathSetARotationAxis

C/C++ Syntax:

```
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

LUA Syntax:

```
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

Description:

Parameters:

Parameter	Description
mInst	The controller instance.

Returns:

Return Code	Description
MERROR_NOERROR	No Error.
MERROR_INVALID_INSTANCE	The mInst parameter was out of range.

Notes:

Usage:

```
//  
MINSTANCE mInst = 0;  
mcToolPathSetARotationAxis(MINSTANCE mInst, int axis);
```

```
int mcToolPathSetAxisColor(  
    MINSTANCE mInst, unsigned long axiscolor, unsigned long limitcolor);  
  
rc = mc.mcToolPathSetAxisColor(  
    number mInst, number axiscolor, number limitcolor)  
  
MINSTANCE mInst = 0;  
  
unsigned long axiscolor;  
  
unsigned long limitcolor;  
  
//red  
  
axiscolor = (255<<0);//red axiscolor += (0<<8);//green axiscolor +=  
(0<<16);//blue  
  
//Yellow  
  
limitcolor = (255<<0);//red limitcolor += (255<<8);//green limitcolor +=  
(128<<16);//blue  
  
<font color="green">// Set the axis and limit colors</font> int rc =  
mcToolPathSetAxisColor(mInst, axiscolor, limitcolor);
```

```
int mcToolPathSetBackColor(
    MINSTANCE mInst, unsigned long topcolor, unsigned long botcolor);
rc = mc.mcToolPathSetBackColor(
    number mInst, number topcolor, number botcolor)
MINSTANCE mInst = 0;
unsigned long topcolor;
```

```
unsigned long botcolor
```

```
//Dark Blue Top
```

```
topcolor = (28<<0); topcolor += (28<<8); topcolor += (213<<16); //Light  
Blue Bottom
```

```
botcolor = (164<<0); botcolor += (156<<8); botcolor += (228<<16);
```

```
<font color="green">// Set the background color</font> int rc =  
mcToolPathSetBackColor(mInst, topcolor, botcolor);
```

```
int mcToolPathSetDrawLimits(  
    MINSTANCE mInst, BOOL drawlimits);  
  
rc = mc.mcToolPathSetDrawLimits(  
    number mInst, number drawlimits)  
  
<font color="green">// Turn on soft limits in the tool path.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcToolPathSetDrawLimits(mInst, TRUE);
```

```
int mcToolPathSetFollowMode(  
    MINSTANCE mInst, BOOL enabled);  
  
rc = mc.mcToolPathSetFollowMode(  
    number mInst, number enabled)  
  
<font color="green">// Turn on tool path jog following.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcToolPathSetFollowMode(mInst, TRUE);
```

```
int mcToolPathSetPathColor(  
    MINSTANCE mInst, unsigned long rapidcolor, unsigned long linecolor,  
    unsigned long arccolor, unsigned long highlightcolor);  
  
rc = mc.mcToolPathSetPathColor(  
    number mInst, number rapidcolor, number linecolor, number arccolor,  
    number highlightcolor)  
  
<font color="green">// Set the tool path colors</font> MINSTANCE mInst  
= 0;  
  
unsigned long rapidcolor;  
  
unsigned long linecolor;  
  
unsigned long arccolor;  
  
unsigned long highlightcolor;  
  
  
  
rapidcolor = (254<<0);//Red rapidcolor += (0<<8);//Green rapidcolor +=  
(0<<16);//Blue  
  
linecolor = (0<<0);//Red  
  
linecolor += (254<<8);//Green linecolor += (0<<16);//Blue  
  
arccolor = (0<<0);//Red  
  
arccolor += (254<<8);//Green arccolor += (0<<16);//Blue  
  
highlightcolor = (254<<0);//Red highlightcolor += (254<<8);//Green  
highlightcolor += (254<<16);//Blue
```

```
int rc = mcToolPathSetPathColor(mInst, rapidcolor, linecolor, arccolor,  
highlightcolor);
```

```
int mcToolPathSetSignalMouseClicks(  
    MINSTANCE mInst, BOOL enabled);  
  
rc = mc.mcToolPathSetSignalMouseClicks(  
    number mInst, number enabled)  
  
<font color="green">// Turn on mouse click events in the tool path.</font>  
MINSTANCE mInst = 0;  
  
int rc = mcToolPathSetSignalMouseClicks(mInst, TRUE);
```

```
int mcToolPathSetView(
```

```
    MINSTANCE mInst, void *parent, int view);
```

N/A

```
<font color="green">// Set the tool path orientation to the ISO view.</font>
MINSTANCE mInst = 0;
```

```
HWND parent; <font color="green">// A valid window handle.</font> int
rc = mcToolPathSetView(mInst, parent, MCTPVIEW_ISO);
```

```
int mcToolPathSetZoom(
```

```
    MINSTANCE mInst, void *parent, double zoom);
```

N/A

```
<font color="green">// Set the tool path zoom percentage to 150%</font>
MINSTANCE mInst = 0;
```

```
HWND parent; <font color="green">// A valid window handle.</font> int
rc = mcToolPathSetZoom(mInst, parent, 1.5);
```

```
int mcToolPathUpdate(  
    MINSTANCE mInst, void *parent);
```

N/A

```
MINSTANCE mInst = 0; HWND parent; <font color="green">// A valid  
window handle.</font> int mcToolPathUpdate(mInst, parent);
```

```
int mcToolSaveFile(  
    MINSTANCE mInst);
```

```
rc = mcToolSaveFile(  
    number mInst)
```

```
<font color="green">// Save the previously loaded tool table file.</font>  
MINSTANCE mInst = 0; int rc = mcToolSaveFile(mInst);
```

```
int mcToolSetCurrent(  
    MINSTANCE mInst, int toolnum);  
  
rc = mc.mcToolSetCurrent(  
    number mInst, number toolnum)  
  
<font color="green">// Set the current tool to tool #1.</font> MINSTANCE  
mInst = 0; int rc = mcToolSetCurrent(mInst, 1);
```

```
int mcToolSetData(  
    MINSTANCE mInst, int Type, int Toolnumber, double val);  
  
rc = mc.mcToolSetData(  
    number mInst, number Type, number Toolnumber, number val);  
  
MINSTANCE mInst = 0; int toolnum = 5;  
  
double val = 0;  
  
<font color="green">// Set the tool height wear offset to zero.</font> int rc  
= mcToolSetData(mInst, MTOOL_MILL_HEIGHT_W, toolnum, val);
```

```
int mcToolSetDataExDbl(  
    MINSTANCE mInst, int toolNum, const char *fieldName, double value);  
  
rc = mc.mcToolSetDataExDbl(  
    number mInst, number toolNum, string fieldName, number val);  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
double val = 0;  
  
<font color="green">// Set the user field "MyUserField" to zero.</font> int  
rc = mcToolSetDataExDbl(mInst, toolnum, "MyUserField", val);
```

```
int mcToolSetDataExInt(  
    MINSTANCE mInst, int toolNum, const char *fieldName, int value);  
  
rc = mc.mcToolSetDataExInt(  
    number mInst, number toolNum, string fieldName, number val);  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
int val = 0;  
  
<font color="green">// Set the user field "MyUserField" to zero.</font> int  
rc = mcToolSetDataExInt(mInst, toolnum, "MyUserField", val);
```

```
int mcToolSetDataExStr(  
    MINSTANCE mInst, int toolNum, const char *fieldName, const char  
    *value);  
  
rc = mc.mcToolSetDataExStr(  
    number mInst, number toolNum, string fieldName, string val);  
  
MINSTANCE mInst = 0;  
  
int toolnum = 5;  
  
char val[128];  
  
<font color="green">// Set the user field "MyUserField" to zero.</font>  
strcpy(val, "0");  
  
int rc = mcToolSetDataExStr(mInst, toolnum, "MyUserField", val);
```

```
int mcToolSetDesc(  
    MINSTANCE mInst, int toolnum, const char *tdsc);  
  
rc = mc.mcToolSetDesc(  
    number mInst, number toolnum, string tdsc);  
  
MINSTANCE mInst = 0;  
  
char desc[128] = "My Best 1/2 inch endmill"; int toolnum = 5;  
  
<font color="green">// Set the description of tool number 5.</font> int rc =  
mcToolSetDesc(mInst, toolnum, desc);
```

Plugin Development

The following topics will discuss things common to all plugins.

- [General Plugin Design Suggestions](#)

The following topics will discuss some of the special function interfaces that are available to a motion plugin.

- [Probing](#)
- [Rigid Tapping](#)
- [Threading](#)



Home



Up a level



Previous



Next

General Plugin Design Suggestions

Registering I/O:

I/O should be registered in the `mcPluginInit()` plugin entry point. It should not depend on the device actually working or in a "connected" state. Instead, I/O configuration should be done from the profile configuration. A "fresh" config could just have a minimal amount of defaults that may be common to the different devices (if the plugin supports multiple devices).

Using this method assures that a device connection failure does not wipe the Core user mappings. Of course you should have a message notifying the user of the connection failure and you should also prevent the plugin from trying to function in this state.

Configuration Dialogs:

Typical config dialogs really should do only two things which are reading and writing settings to the profile. The plugin should read the settings from the profile upon startup and after a config change. Meaning it is usually not a good idea to pass pointers to control structures or classes from the plugin and have the configuration dialog modify them. It is much better to isolate the config dialog from the plugin in this manner. You should create a plugin config dialog with the goal of it being able to work autonomously to the extent where the dialog could be used in a separate config executable.

Detecting signal, I/O, and register changes:

The Core uses a messaging system to keep the plugins informed of signal and register changes. It is not necessary to poll signal states or registers at all. If the Core changes an output signal, it sends a MSG_SIG_CHANGED message to the plugin. All you have to do is watch for any given signal you are interested in. The signal ID will be passed in param1 and the state passed in param2. The same for registers except your plugin will only receive register changes for registers that your plugin created via the mcPluginMsg() (synchronous) callback. The register handle will be passed in param 1. You must then call mcRegisterGet*() functions to retrieve the new value of the register since they can contain any type of data like strings or double values that cannot be passed in param2.

It is also not necessary to poll the state of outputs that your plugin created. If a user has mapped them to a Core signal, and the Core changes the state of the output via changing its mapped signal, then your plugin will get a MSG_IO_CHANGED message via the mcPluginMsg() callback. The handle to the IO object will be passed in param1 and the state will be passed in param2.

Detecting configuration changes:

The Core sends a MSG_CONFIG_END to the plugins at startup when all of the device, axis, and motor information is available. This is the point at which the Core has everything loaded and configured and is ready for operation. If the user enters the Core configuration dialog, The Core will send a MSG_CONFIG_START. It will also send a MSG_CONFIG_END when the user is finished the configuring process. You can use the MSG_CONFIG_END message in your plugin to do any housekeeping that needs to be done after configuration changes. If you see a MSG_CONFIG_START, you can put your device into a benign state, if you wish, but MSG_CONFIG_END must take it out of this state.

Controlling the interoperability with other Core devices:

Every now and then, it may be desirable to have your plugin only work if other devices are present. Similarly, you may wish to disable other devices that you deem necessary. As an example, if you wanted the Core to only run with your plugin, then in your plugin, you can loop through all of the Core devices and disable any other motion device (DEV_TYPE_MOTION flag) it finds. You might want to leave Sim operational though.



Home



Up a level



Previous



Next

Probing

Motion plugin probing procedure.

The probe moves are really just **G01** moves in exact stop mode with the addition of being marked as **EX_PROBE** in the **execution_t** struct from

[mcMotionCyclePlannerEx\(\)](#).

When executing a **G31**, the core considers the move as an exact stop move and therefore will request a motor stop report. It waits on one of two conditions:

1. A probe strike condition where the motion plugin calls

[mcMotionSetProbeComplete\(\)](#).

2. An end of move condition where the motion plugin reports the motors as being still.

Until either of these conditions are satisfied, the core will generate no more moves after the **EX_PROBE** marked moves. In other words, all you will get out of

[mcMotionCyclePlannerEx\(\)](#) will be **EX_NONE** type data. In

the event of a probe strike, this makes it safe to clear the planner and be sure that future moves are not removed by accident.

A motion plugin should implement probing in the following way:

1. Upon seeing the first move marked as **EX_PROBE**, the plugin should arm position latches on the hardware.
2. Then consume the **EX_PROBE** marked moves as normal.
3. If the probe stikes, the plugin should:
 - Cancel any motor stop report requests (retrieved from the `execution_t::move_t::exMotors` structure). (see below. VERY important!)
 - Report the latched positions via

[mcMotionSetProbePos\(\)](#) for each motor.

- Clear the hardware of any additional moves.
- Call

[mcMotionClearPlanner\(\)](#) to clear the Mach planner of any unretrieved **EX_PROBE** moves.

- Update Mach with the hardware's current position via

[mcMotionSetPos\(\)](#).

- Call

[mcMotionSetProbeComplete\(\)](#).

- If any motor stop report requests has been received, do not acknowledge them.

[mcMotionSetProbeComplete\(\)](#) does this for you and a position sync as well.

4. If no probe strike, then the move continues to it's end point as if it is a regular move.

- The plugin should ack any motor stop report requests with

[mcMotionSetStill\(\)](#).

- The positions latches should be disarmed (if needed) when a **MSG_PROBE_DONE** is seen.

5. In the event that the probe move is aborted, the plugin should:

- Call

[mcMotionClearPlanner\(\)](#) to clear the Mach planner of any unretrieved **EX_PROBE** moves.

- Call

[mcMotorSetPos\(\)](#) for each controlled motor.

- Call

[mcMotionSync](#).

- Call

[mcMotionSetStill\(\)](#) for each controlled motor.

Probe Operation Overview.

1. Upon executing G31, the core will mark those moves as **EX_PROBE** in the data stream.
2. If the probe strikes before the end of the G31 move, the plugin aborts the rest of

the probe moves as described above. If a probe data file has been opened via

[mcCtlProbeFileOpen\(\)](#), then the positions recorded by the motion plugin are inserted

into the probe data file in the specified format.

3. If the probe doesn't strike and the G31 move reaches its end point and a probe

data file has been opened via

[mcCtlProbeFileOpen\(\)](#), then the end point positions

are inserted into the probe data file in the specified format.

Using the probe data file routines.

```
int mcCtlProbeFileOpen(MINSTANCE inst, const char *filename,  
const char *format);
```

The format argument is a printf style format with expanding macros for the axis

values. **AXIS_X**, **AXIS_Y**, **AXIS_Z**, **AXIS_A**, **AXIS_B**, **AXIS_C**. It is used in the

following manner:

```
mcCtlProbeFileOpen(inst, "myProbeFile.csv", "%.<4>AXIS_X,  
%.4AXIS_Y, %.4AXIS_Z");
```

This will produce a probe file in CSV format like so:

```
1.0000, 2.0000, -1.4356  
1.0100, 2.0000, -1.4343  
1.0200, 2.0000, -1.4324  
...
```

A format of "**X%.4AXIS_X**, **Y%.4AXIS_Y**, **Z%.4AXIS_Z**" would yield:

```
X1.0000, Y2.0000, Z-1.4356  
X1.0100, Y2.0000, Z-1.4343  
X1.0200, Y2.0000, Z-1.4324  
...
```

A format of "X%.4AXIS_X\tY%.4AXIS_Y\tZ%.4AXIS_Z" would yield a tab delimited file:

```
X1.0000 Y2.0000 Z-1.4356
X1.0100 Y2.0000 Z-1.4343
X1.0200 Y2.0000 Z-1.4324
...
```

```
int mcCtlProbeFileClose(MINSTANCE inst); Closes the probe data
file.
```



Home



Up a level



Previous



Next

Rigid Tapping

Motion plugin rigid tapping procedure.

A plugin must use either one of two rigid tapping interfaces. (Or none is rigid tapping is not supported.) This interface is defined when the motion plugin registers its motion device with

[mcDeviceRegister\(\)](#) using the **Type** parameter bitwise values of **DEV_TYPE_RTAP** or **DEV_TYPE_RTAP2**.

- **DEV_TYPE_RTAP:** This interface is defined if the motion controller is capable of precisely controlling the spindle speed with regard to the Z movement profile. The core profiles the Z moves and they are output with **EX_RTAP** movement types in the **execution_t** struct from [mcMotionCyclePlannerEx\(\)](#).
- **DEV_TYPE_RTAP2:** This interface is defined if the motion controller wants to handle the complete coordination of the rigid tap move. With this interface, no Z move is profiled by the core. Instead, the complete move points are provided in the **tap_t** structure.

Using the DEV_TYPE_RTAP2 interface

When **DEV_TYPE_RTAP2** is selected, the core will output one place holder move with **EX_RTAP** from

[mcMotionCyclePlannerEx\(\)](#). When this is seen by the plugin, it should:

1. Retrieve the **tap_t** struct information with
[mcMotionGetRigidTapParams\(\)](#).
2. Profile the moves and synchronize the spindle according to the **tap_t** struct information
3. Acknowledge any **MSG_REPORT_MOTOR_STOP** messages with
[mcMotionSetStill\(\)](#).

 Home

 Up a level

 Previous

Threading

Motion plugin threading procedure.

A plugin must use either one of two threading interfaces. (Or none if threading is not supported.) This interface is defined when the motion plugin registers its motion device with

[mcDeviceRegister\(\)](#) using the **Type** parameter bitwise values of **DEV_TYPE_THREAD** or **DEV_TYPE_THREAD2**.

- **DEV_TYPE_THREAD:** This interface is defined if the motion controller is capable of shrinking or expanding the time of the way points in regard to the spindle speed. The core profiles the moves and they are output with **EX_THREAD** movement types in the **execution_t** struct from

[mcMotionCyclePlannerEx\(\)](#). Only the **G32** moves are marked with **EX_THREAD**.

The challenge is to derive a ratio from the programmed spindle speed and the actual spindle speed. If the speed is slow by 1%, then the way point timeslice time should be increased by 1%, etc... The higher the frequency that the actual spindle speed can be read, the greater the thread accuracy.

The entry and exit moves from a **G76** are considered "normal" movement types and do not require any spindle synchronization.

- **DEV_TYPE_THREAD2:** This interface is defined if the motion controller wants to handle the complete coordination of the threading tap move. With this interface, no moves are profiled by the core. Instead, the complete move points are provided in the **tap_t** structure.

Using the **DEV_TYPE_THREAD2** interface (Not implemented currently)

When **DEV_TYPE_THREAD2** is selected, the core will send a synchronous **MSG_THREAD_START** message. When this is seen by the plugin, it should:

1. Retrieve the **thread_t** struct information with
[mcMotionGetThreadParams\(\)](#).
2. Profile the moves and synchronize the spindle according to the **thread_t** struct information
3. Acknowledge any **MSG_REPORT_MOTOR_STOP** messages with
[mcMotionSetStill\(\)](#).