API Reference

2019/11/08

Note:

- all functions use CDECL calling convention.

- some commands can be called using named pipe \\.\pipe\PlanetCNC

- use one of Run commands to start new TNG process. If API is calling existing TNG process then it will internally use pipes for interprocess communication.

Memory

FreeString

Declaration: `void FreeString(char* str);`

Frees previously allocated string.

FreeStringW

Declaration: `void FreeStringW(wchar_t* strw);`

Frees previously allocated widestring.

Callback

SetInitialiseCB

Declaration: `void SetInitialiseCB(__InitialiseCB cb);`

Callback is called when TNG is initialed and ready to use (parameter value '1') and when TNG is shutdown and no longer available (parameter value '0').

Prototype `__InitialiseCB` is defined as:
`typedef void(* __InitialiseCB)(int);`

SetRefreshCB

Declaration: `void SetRefreshCB(__RefreshCB cb);`

Callback is called when TNG is refreshed. This callback is called with high frequency and should be used with caution.

Prototype `__RefreshCB` is defined as:
`typedef void(* __RefreshCB)();`

SetOpenCB

Declaration: `void SetOpenCB(__OpenCB cb);`

Callback is called when TNG opened a file. Parameter value '1' indicates that file was successfully opened.

Prototype `__OpenCB` is defined as:
`typedef void(* __OpenCB)(int);`

SetOutputCB

Declaration: `void SetOutputCB(__OutputCB cb);`

Callback is called when string is send to output window.

Prototype `__OutputCB` is defined as:
`typedef void(* __OutputCB)(const char*);`

SetOutputWCB

Declaration: `void SetOutputWCB(__OutputWCB cb);`

Callback is called when string is send to output window.

Prototype __OutputWCB is defined as:
`typedef void(* __OutputWCB)(const wchar_t*);`

SetIdleCB

Declaration: `void SetIdleCB(__IdleCB cb);`

Callback is called when controller completed task and is back to idle.

Prototype __IdleCB is defined as:
`typedef void(* __IdleCB)();`

Run & Exit

Run

Declaration: `int Run(bool hideUI);`

Runs new TNG instance with visible or hidden user interface. TNG process will run in same process as calling application. This is fastest and preferred way to use API.

RunProfile

Declaration: `int RunProfile(bool hideUI, const char* profile);`

Runs new TNG instance with specified profile loaded with visible or hidden user interface. TNG process will run in same process as calling application and you can also specify which profile to use.

RunProfileW

Declaration: `int RunProfileW(bool hideUI, const wchar_t* profile);`

Same as 'RunProfile' except widestring is used for profile name.

Exit

Declaration: `void Exit();`
Pipe: *exit*

Exits TNG instance if possible.

ExitForce

Declaration: `void ExitForce();`

Exits TNG instance.

Run status

GetVer

Declaration: `int GetVer();`

Returns TNG version number. This command is usually first to call and is used to check if TNG library is available to API. If you get error or zero then something is wrong and TNG API can not be used.

GetLibPath

Declaration: `char* GetLibPath();`

Note: Use FreeString to free returned string.

Returns TNG library path.

GetLibPathW

Declaration: `wchar_t* GetLibPathW();`

Note: Use FreeStringW to free returned widestring.

Returns TNG library path.

IsRunning

Declaration: `bool IsRunning();`

Returns 'true' if TNG is running in same process as calling application. This is fastest and preferred way.

IsRunningExt

Declaration: `bool IsRunningExt();`

Returns 'true' if TNG is running as external process. In this case interprocess communication is made with named pipes. Not all commands are available. This is usually used for simple applications that exchange only limited amount of data.

IsInitialized

Declaration: `bool IsInitialized();`

Returns 'true' if TNG is running in-process and initialization is completed.

Version & Profile Info

GetVersionString

Declaration: `const char* GetVersionString();`
Pipe: *version*

Note: Do not use `FreeString` on this function.

Returns version string.

GetVersionStringW

Declaration: `const wchar_t* GetVersionStringW();`

Note: Do not use `FreeStringW` on this function.

Returns version widestring.

GetProfileNameString

Declaration: `const char* GetProfileNameString();`
Pipe: *profname*

Note: Do not use `FreeString` on this function.

Returns profile name string.

GetProfileNameStringW

Declaration: `const wchar_t* GetProfileNameStringW();`

Note: Do not use `FreeStringW` on this function.

Returns profile name widestring.

GetDescriptionString

Declaration: `const char* GetDescriptionString();`
Pipe: *description*

Note: Do not use `FreeString` on this function.

Returns profile description string.

GetDescriptionStringW

Declaration: `const wchar_t* GetDescriptionStringW();`

Note: Do not use `FreeStringW` on this function.

Returns profile description widestring.

Screen

IsVisible

Declaration: `bool IsVisible();`
Pipe: *isvisible*

Returns 'true' if TNG main window is visible.

Show

Declaration: `bool Show();`
Pipe: *show*

Makes TNG main window visible.

Hide

Declaration: `bool Hide();`
Pipe: *hide*

Makes TNG main window hidden.

SetProgress

Declaration: `bool SetProgress(double dbl);`
Pipe: *setprogress <value>*

Sets progress bar value.

SetStatus

Declaration: `bool SetStatus(const char* str);`
Pipe: *setstatus <text>*

Sets status bar text.

SetStatusW

Declaration: `bool SetStatusW(const wchar_t* strw);`

Sets status bar text.

Msg

Declaration: `bool Msg(const char* str);`
Pipe: *msg <text>*

Shows message dialog.

MsgW

Declaration: `bool MsgW(const wchar_t* strw);`

Shows message dialog.

Print

Declaration: `bool Print(const char* str);`
Pipe: *print <text>*

Prints text to output window.

PrintW

Declaration: `bool PrintW(const wchar_t* strw);`

Prints text to output window.

Parameters & Eval

GetParam

Declaration: `double GetParam(const char* param);`
Pipe: *param <name>*

Returns parameter value.

GetParamW

Declaration: `double GetParamW(const wchar_t* param);`

Returns parameter value.

SetParam

Declaration: `bool SetParam(const char* param, double value);`
Pipe: *param <name>=<value>*

Sets parameter value.

SetParamW

Declaration: `bool SetParamW(const wchar_t* param, double value);`

Sets parameter value.

Evaluate

Declaration: `double Evaluate(const char* expr);`
Pipe: *evaluate <expr>*

Evaluates expression.

EvaluateW

Declaration: `double EvaluateW(const wchar_t* expr);`

Evaluates expression.

State

IsLicenseValid

Declaration: `bool IsLicenseValid();`
Pipe: *islicensevalid*

Returns 'true' if license is valid.

IsUIReady

Declaration: `bool IsUIReady();`
Pipe: *isuiready*

Returns 'true' if user interface is ready and not locked. During some commands or when dialog os shown UI might be locked.

IsControllerReady

Declaration: `bool IsControllerReady();`
Pipe: *iscontrollerready*

Returns 'true' if controller is ready.

IsControllerRunning

Declaration: `bool IsControllerRunning();`
Pipe: *iscontrollerrunning*

Returns 'true' if controller is running.

IsProgramLoaded

Declaration: `bool IsProgramLoaded();`
Pipe: *isprogramloaded*

Returns 'true' if g-code program is loaded.

IsIdle

Declaration: `bool IsIdle();`
Pipe: *isidle*

Returns 'true' if motion engine is idle.

IsEStop

Declaration: `bool IsEStop();`
Pipe: *isestop*

Returns 'true' if e-stop is active.

IsStop

Declaration: `bool IsStop();`
Pipe: *isstop*

Returns 'true' if controller is stopped.

IsPause

Declaration: `bool IsPause();`
Pipe: *ispause*

Returns 'true' if controller is paused.

Machine Command Status

IsEStopEnabled

Declaration: `bool IsEStopEnabled();`
Pipe: *isestopenabled*

Returns 'true' if e-stop command is enabled.

IsStopEnabled

Declaration: `bool IsStopEnabled();`
Pipe: *isstopenabled*

Returns 'true' if stop command is enabled.

IsPauseEnabled

Declaration: `bool IsPauseEnabled();`
Pipe: *ispauseenabled*

Returns 'true' if pause command is enabled.

IsStartEnabled

Declaration: `bool IsStartEnabled();`
Pipe: *isstartenabled*

Returns 'true' if start commands are enabled.

IsOutputEnabled

Declaration: `bool IsOutputEnabled();`
Pipe: *isoutputenabled*

Returns 'true' if output commands are enabled.

IsCodeEnabled

Declaration: `bool IsCodeEnabled();`
Pipe: *iscodeenabled*

Returns 'true' if other commands are enabled.

IsCodeExEnabled

Declaration: `bool IsCodeExEnabled();`
Pipe: *iscodeexenabled*

Returns 'true' if special commands are enabled (home, measure,...).

IsOpenEnabled

Declaration: `bool IsOpenEnabled();`
Pipe: *isopenenabled*

Returns 'true' if open commands are enabled.

IsCloseEnabled

Declaration: `bool IsCloseEnabled();`
Pipe: *iscloseenabled*

Returns 'true' if close command is enabled.

Machine Commands

EStop

Declaration: `bool EStop(bool b);`
Pipe: *estop_enable*
Pipe: *estop_disable*

Executes 'estop' command.

EStopToggle

Declaration: `bool EStopToggle();`
Pipe: *estop*

Executes 'estop' command.

Stop

Declaration: `bool Stop();`
Pipe: *stop*

Executes 'stop' command.

Pause

Declaration: `bool Pause(bool b);`
Pipe: *pause_enable*
Pipe: *pause_disable*

Executes 'pause' command.

PauseToggle

Declaration: `bool PauseToggle();`
Pipe: *pause*

Executes 'pause' command.

Start

Declaration: `bool Start();`
Pipe: *start*

Executes 'start' command.

Open

Declaration: `bool Open();`
Pipe: *open*

Executes 'open' command.

Close

Declaration: `bool Close();`
Pipe: *close*

Executes 'close' command.

Machine Commands Generic

GetCmdCount

Declaration: `int GetCmdCount();`
Pipe: *getcmdcount*

Returns command count.

GetCmdId

Declaration: `int GetCmdId(const char* str);`
Pipe: *getcmdid*

Returns command id from command name.

GetCmdIdW

Declaration: `int GetCmdIdW(const wchar_t* strw);`

Returns command id from command name.

GetCmdIdFromMdi

Declaration: `int GetCmdIdFromMdi(const char* str);`
Pipe: *getcmdidmdi*

Returns command id from MDI shortcut.

GetCmdIdFromMdiW

Declaration: `int GetCmdIdFromMdiW(const wchar_t* strw);`

Returns command id from MDI shortcut.

IsCmdEnabled

Declaration: `IsCmdEnabled(int id);`
Pipe: *iscmdenabled*

Returns 'true' if command is enabled.

IsCmdChecked

Declaration: `bool IsCmdChecked(int id);`
Pipe: *iscmdchecked*

Returns 'true' if command is activated.

GetCmdName

Declaration: `char* GetCmdName(int id);`
Pipe: *getcmdname*

Note: Use FreeString to free returned string.

Returns command name string.

GetCmdNameW

Declaration: `wchar_t* GetCmdNameW(int id);`

Note: Use FreeStringW to free returned widestring.

Returns command name widestring.

GetCmdDisplayName

Declaration: `char* GetCmdDisplayName(int id);`
Pipe: *getcmddispname*

Note: Use FreeString to free returned string.

Returns command display name string.

GetCmdDisplayNameW

Declaration: `wchar_t* GetCmdDisplayNameW(int id);`

Note: Use FreeStringW to free returned widestring.

Returns command display name widestring.

GetCmdDisplayNameWithPath

Declaration: `char* GetCmdDisplayNameWithPath(int id);`

Pipe: *getcmddispnamepath*

Note: Use FreeString to free returned string.

Returns command display name string with full path.

GetCmdDisplayNameWithPathW

Declaration: `wchar_t* GetCmdDisplayNameWithPathW(int id);`

Note: Use FreeStringW to free returned widestring.

Returns command display name widestring with full path.

CmdExec

Declaration: `bool CmdExec(int id);`
Pipe: *cmdexec*

Executes command.

CmdExecStr

Declaration: `bool CmdExecStr(int id, const char* str);`
Pipe: *cmdexecstr*

Executes command with string parameter.

CmdExecStrW

Declaration: `bool CmdExecStrW(int id, const wchar_t* strw);`

Executes command with widestring parameter.

CmdExecVal

Declaration: `bool CmdExecVal(int id, double val);`
Pipe: *cmdexecval*

Executes command with double value parameter.

CmdExecVal2

Declaration: `bool CmdExecVal2(int id, int num, double val);`
Pipe: *cmdexecval2*

Executes command with integer and double value parameter.

Helpers

OpenFn

Declaration: `bool OpenFn(const char* str);`
Pipe: *openfile*

Opens g-code file.

OpenFnW

Declaration: `bool OpenFnW(const wchar_t* strw);`

Opens g-code file.

OpenCode

Declaration: `bool OpenCode(const char* str);`
Pipe: *opencode*

Opens g-code from string.

OpenCodeW

Declaration: `bool OpenCodeW(const wchar_t* strw);`

Opens g-code from widestring.

TestFn

Declaration: `int TestFn(const char* str);`
Pipe: *testfile*

Tests g-code file.

TestFnW

Declaration: `int TestFnW(const wchar_t* strw);`

Tests g-code file.

TestCode

Declaration: `int TestCode(const char* str);`
Pipe: *testcode*

Tests g-code from string.

TestCodeW

Declaration: `int TestCodeW(const wchar_t* strw);`

Tests g-code from widestring.

StartFn

Declaration: `bool StartFn(const char* str);`
Pipe: *startfile*

Starts g-code file.

StartFnW

Declaration: `bool StartFnW(const wchar_t* strw);`

Starts g-code file.

StartCode

Declaration: `bool StartCode(const char* str);`
Pipe: *startcode*

Starts g-code string.

StartCodeW

Declaration: `bool StartCodeW(const wchar_t* strw);`

Starts g-code widestring.

G-Code

GetFn

Declaration: `char* GetFn();`
Pipe: *filename*

Note: Use FreeString to free returned string.
This command can not be used if TNG is external process.

Returns opened g-code filename.

GetFnW

Declaration: `wchar_t* GetFnW();`

Note: Use FreeStringW to free returned widestring.
This command can not be used if TNG is external process.

Returns opened g-code filename.

GetLineCount

Declaration: `int GetLineCount();`
Pipe: *linecount*

Returns opened g-code line count.

GetLine

Declaration: `char* GetLine(int row);`
Pipe: *line <row>*

Note: Use FreeString to free returned string.
This command can not be used if TNG is external process.

Returns opened g-code row string.

GetLineW

Declaration: `wchar_t* GetLineW(int row);`

Note: Use FreeStringW to free returned widestring.
This command can not be used if TNG is external process.

Returns opened g-code row widestring.

GetLineNum

Declaration: `int GetLineNum();`
Pipe: *linenum*

Returns opened g-code current line number.

Info

InfoIsInitialized

Declaration: `bool InfoIsInitialized();`
Pipe: *isinitialized*

Returns 'true' if TNG is initialized.

InfoSerial

Declaration: `int InfoSerial();`
Pipe: *serial*

Returns controller serial number.

InfoHWVersion

Declaration: `int InfoHWVersion();`
Pipe: *hwversion*

Returns controller version.

InfoSWVersion

Declaration: `int InfoSWVersion();`
Pipe: *swversion*

Returns software version.

InfoIsVersionValid

Declaration: `bool InfoIsVersionValid();`
Pipe: *versionvalid*

Returns 'true' if controller version is valid for this software version.

InfoWorkPosition

Declaration: `double InfoWorkPosition(int n);`
Pipe: *posworkx – posworkw*

Returns work position for axis (0-8).

InfoWorkPosition3

Declaration: `bool InfoWorkPosition3(double* x, double* y, double* z);`

Returns X, Y and Z work position.

InfoWorkPosition9

Declaration: `bool InfoWorkPosition9(double* x, double* y, double* z,`
`                        double* a, double* b, double* c,`
`                        double* u, double* v, double* w);`

Pipe: *poswork*

Returns work position for all axes.

InfoMotorPosition

Declaration: `double InfoMotorPosition(int n);`
Pipe: *posmotorx - posmotorw*

Returns motor position for axis (0-8).

InfoMotorPosition3

Declaration: `bool InfoMotorPosition3(double* x, double* y, double* z);`

Returns X, Y and Z motor position.

InfoMotorPosition9

Declaration: `bool InfoMotorPosition9(double* x, double* y, double* z,`
`                                     double* a, double* b, double* c,`
`                                     double* u, double* v, double* w);`

Pipe: *posmotor*

Returns motor position for all axes.

`InfoWorkPosition`

Declaration: `double InfoWorkPosition(int n);`
Pipe: *posworkx – posworkw*

Returns work position for axis (0-8).

InfoWorkUnitsPosition

Declaration: `double InfoWorkUnitsPosition(int n);`
Pipe: *posworkunitsx – posworkunitsw*

Returns work position for axis (0-8) in user units.

InfoWorkUnitsPosition3

Declaration: `bool InfoWorkUnitsPosition3(double* x, double* y, double* z);`

Returns X, Y and Z work position in user units.

InfoWorkUnitsPosition9

Declaration: `bool InfoWorkUnitsPosition9(double* x, double* y, double* z,`
`                                    double* a, double* b, double* c,`
`                                    double* u, double* v, double* w);`

Pipe: *posworkunits*

Returns work position for all axes in user units.

InfoMotorUnitsPosition

Declaration: `double InfoMotorUnitsPosition(int n);`
Pipe: *posmotorunitsx - posmotorunitsw*

Returns motor position for axis (0-8) in user units.

InfoMotorUnitsPosition3

Declaration: `bool InfoMotorUnitsPosition3(double* x, double* y, double* z);`

Returns X, Y and Z motor position in user units.

InfoMotorUnitsPosition9

Declaration: `bool InfoMotorUnitsPosition9(double* x, double* y, double* z,`
`double* a, double* b, double* c,`
`double* u, double* v, double* w);`

Pipe: *posmotorunits*

Returns motor position for all axes in user units.

InfoSpeed

Declaration: `double InfoSpeed();`
Pipe: *speed*

Returns current speed.

InfoAcceleration

Declaration: `double InfoAcceleration();`
Pipe: *accel*

Returns current acceleration.

InfoSpindle

Declaration: `double InfoSpindle();`
Pipe: *spindle*

Returns current spindle speed.

InfoSpindleIdx

Declaration: `double InfoSpindleIdx();`
Pipe: *spindle_idx*

Returns current spindle speed read from idx signal.

InfoSpindleEnc

Declaration: `double InfoSpindleEnc();`
Pipe: *spindle_enc*

Returns current spindle speed read from spindle encoder

InfoSpindleSet

Declaration: `double InfoSpindleSet();`
Pipe: *spindle_set*

Returns current spindle speed as set by g-code

InfoSpeedOverrideEnabled

Declaration: `bool InfoSpeedOverrideEnabled();`
Pipe: *speed_ovrden*

Returns 'true' if speed override is enabled.

InfoSpindleOverrideEnabled

Declaration: `bool InfoSpindleOverrideEnabled();`
Pipe: *spindle_ovrden*

Returns 'true' if spindle override is enabled.

InfoInput

Declaration: `unsigned int InfoInput();`
Pipe: *input*

Returns INPUT connector pin state.

InfoJog

Declaration: `unsigned int InfoJog();`
Pipe: *jog*

Returns JOG connector pin state.

InfoJogPot

Declaration: `unsigned int InfoJogPot();`
Pipe: *jogpot*

Returns JOG potentiometer value.

InfoLimit

Declaration: `unsigned int InfoLimit();`
Pipe: *limit*

Returns LIMIT connector pin state.

InfoOutput

Declaration: `unsigned int InfoOutput();`
Pipe: *output*

Returns OUTPUT connector pin state.

InfoOutputFreq

Declaration: `double InfoOutputFreq(int num);`
Pipe: *outputfreq1 - outputfreq3*

Returns OUTPUT connector PWM frequency.

InfoOutputDuty

Declaration: `double InfoOutputDuty(int num);`
Pipe: *outputduty1 - outputduty3*

Returns OUTPUT connector PWM duty cycle.

InfoAux

Declaration: `unsigned int InfoAux(int num);`

Pipe: *aux1 – aux4*

Returns AUX connector pin state. Pins Aux2 and Aux4 are analog pins.

InfoBufferAvailable

Declaration: `unsigned int InfoBufferAvailable();`

Pipe: *buffer*

Returns 'true' if controller has available buffer.

InfoBufferUtilization

Declaration: `double InfoBufferUtilization();`

Pipe: *bufferutil*

Returns controller buffer utilization percent.

Direct Commands

MoveAxis

Declaration: `bool MoveAxis(double speed, int axis, double val);`
Moves axis to position.

Move3

Declaration: `bool Move3(double speed, double x, double y, double z);`
Moves axes X, Y and Z to position.

Move9

Declaration: `bool Move9(double speed, double x, double y, double z,`
`                         double a, double b, double c,`
`                         double u, double v, double w);`

Moves all axes to position.

Output

Declaration: `bool Output(int num, bool value);`
Turns OUTPUT pin on/off.

OutputPWM

Declaration: `bool OutputPWM(int num, int freq, double duty);`
Sets OUTPUT pin PWM state.

OutputRC

Declaration: `bool OutputRC(int num, double value);`
Sets OUTPUT pin RC servo state.

I2C

Declaration: `bool I2C(int addr, const unsigned char* data, int len);`
Sends I2C command to external peripheral.

I2Cret

Declaration: `bool I2Cret(int addr, const unsigned char* data, int len, unsigned char* dataret, int lenret);`

Sends I2C command to external peripheral and returns value.

Jog

JogStop

Declaration: `bool JogStop();`

Stops jogging.

Jog

Declaration: `bool Jog(bool step, double x, double y, double z);`

Start or continue jogging axes X, Y and Z.

Jog9

Declaration: `bool Jog9(bool step, double x, double y, double z,`
`                       double a, double b, double c,`
`                       double u, double v, double w);`

Start or continue jogging all 9 axes.

G-Code Lines

LineListCreate

Declaration: `void* LineListCreate();`

LineListFree

Declaration: `void LineListFree(void* ptr);`

LineListClear

Declaration: `bool LineListClear(void* ptr);`

LineListCount

Declaration: `int LineListCount(void* ptr);`

LineListAddAllowed

Declaration: `bool LineListAddAllowed(void* ptr);`

LineListAdd

Declaration: `bool LineListAdd(void* ptr, const char* str);`

LineListAddW

Declaration: `bool LineListAddW(void* ptr, const wchar_t* str);`

LineListGet

Declaration: `char* LineListGet(void* ptr, int i);`

LineListGetW

Declaration: `wchar_t* LineListGetW(void* ptr, int i);`

LineListLoadFromFile

Declaration: `bool LineListLoadFromFile(void* ptr, const char* fn);`

LineListLoadFromFileW

Declaration: `bool LineListLoadFromFileW(void* ptr, const wchar_t* fn);`

OpenLineList

Declaration: `bool OpenLineList(void* ptr);`

TestLineList

Declaration: `int TestLineList(void* ptr);`

StartLineList

Declaration: `bool StartLineList(void* ptr);`

Table of Contents