# TCP Command Server (TCS)
# User's Guide

Version 3.0C1, May 2020

## 1. Overview

The TCP Command Server (TCS) is an application that is written in Precise Automation's GPL (Guidance Programming Language) that allows remote commands to be executed using simple string messages over TCP/IP Ethernet. An alternate configuration allows commands over standard serial lines.

All of the basic TCS is open source which allows end users to alter it to a specific application using the Precise Guidance Development Environment (GDE).

Once TCS is installed and configured, the Precise controller acts as a command server that supports up to 8 robot devices. Each device can have up to 12 axes of motion. Each of these robots runs independently of the others and uses a separate communication channel. Up to 4 additional channels not associated with any robot can be used to monitor status or perform digital I/O operations.

### 1.1. Version 3.0 Changes and Compatibility

For details about the changes in each release, please read the file "Release-notes.txt" found in the distribution.

Version 3.0 of TCS is generally not compatible with older TCS versions. The following non-comprehensive list of changes have been made:

1. The concept of current location and current profile has been removed. You must explicitly specify the location and profile index in any command that requires them. That means the "locidx" and "profidx" commands have been removed and all the commands that referenced the current index now require an index parameter.
2. All profile and location index values now start at 1. Attempts to use 0 generate an error.
3. The location arrays have been replaced by arrays of "Station" class objects. The terms "location index", "station index" and "station ID" in these documents are interchangeable and refer to the same object.
4. Pallets are now supported for Cartesian locations. See the pallet commands in the PARobot plug-in. The "loc" command shows the location including any pallet offset.
5. The tcs.gpo file format has changed. You cannot move .gpo files between version 3.0 and older versions. The .gpo files now contain complete station information.

# 2. Quick Start

## 2.1. System Requirements

The controller must be running in GPL mode and have GPL OS version 3.2 or higher. Some plug-ins may require a higher version of GPL. If you plan to customize TCS, you will need the Guidance Development Environment (GDE), which is part of the Guidance Development Suite (GDS). We recommend version 4.0.0.2 or higher.

## 2.2. Installation

TCS is distributed as GPL project named "Tcp_cmd_server". The program files are found in a folder with that name. Most users will want to copy this project to the Precise controller flash disk. You can use an FTP client to connect to your controller and copy the Tcp_cmd_server folder into your controller's /flash/projects/folder. Alternatively, you can use GDE or the Guidance Update Wizard (which is also part of the GDS package) to copy the project.

## 2.3. Startup

When loaded in flash, TCS may be started from the operator control panel or may be auto-started when the controller boots (see the web page Setup > System Setup > Wizards and Setup Tools > Startup Configuration).

## 2.4. Connections

### 2.4.1. TCP Ethernet Connections

By default, TCS listens on two TCP ports for a client to connect. You can connect using a Telnet client for testing. From Telnet, specify your controller's IP address and port 10000 for the first status connection or 10100 for robot 1. If you enable additional robots from TCP, they use ports 10n00 where *n* is the robot number. If you enable additional status connections, they use ports 10001 through 10003.

Since TCP/IP is a stream connection, the host cannot assume that there is a 1-to-1 correspondence between TCP/IP packets and commands or replies. A command or reply may be split into multiple TCP/IP packets, or a single packet may contain fragments of multiple commands or replies. For reliable operation, the host interface software must handle these cases.

### 2.4.2. Serial Connections

Changing a configuration variable (see section *Core Program Customization* below), causes TCS to communicate over an RS-232 serial port instead of Ethernet. In serial mode, each command starts with a single number that indicates the robot to which the command is directed. The number 0 directs the command to the status thread. Only one status thread is supported for serial connections. Serial verbose mode may optionally be set to echo received characters.

### 2.4.3. Command and Reply

Once connected, TCS reads commands, processes them, and sends a reply. For example, the command "version" will display the TCS version information. The command "exit" closes the TCP connection.

There are two modes that TCS runs in. One provides verbose responses (used when using Telnet) and the other is PC mode where the responses are designed to be easily parsed by a PC based TCP client application. At startup, TCS is in PC mode. To enter verbose mode, enter the command "mode 1". To return to PC mode, enter the command "mode 0".

## 2.5. Commands

TCS sends a reply for each command (except for "exit"). In PC mode, each response begins with a number that indicates the success or failure of the command. The value "0" indicates success. A negative value corresponds to a GPL or TCS error code and indicates failure of that command. Details on the standard commands are in a section below.

## 2.6. Shutdown

Once TCS starts, it continues to listen on its TCP ports until it is shutdown explicitly from the web interface or GDE.

## 2.7. Plug-Ins

TCS is designed to allow additional commands to be added without the need to modify the core TCS program files. To add a plug-in to TCS with GDE, drag the plug-in .gpl file into the Tcp_cmd_server project. Then reload the project. TCS should automatically detect the plug-in.

Some versions of TCS may include plug-ins that are not required for your application. To remove them, delete their files from the project.

Details on using and creating plug-ins are found in a section below.

# 3. General Concepts

## 3.1. Stations

Stations are places to which the robot can move. From within TCS, by default, you can access up to 20 stations, numbered 1 to 20. This limit can be easily increased for your application. The stations are referenced by the *station index*, from 1 to n, that is specified as a parameter to many commands. Stations contain a robot location, rail position, and pallet information.

The robot locations can be of type *angles* or *Cartesian* as described below.

### 3.1.1. Angles Robot Locations

An angles location is a collection of the joint angles for the robot. Joint angles are in units of degrees for rotary axes and in millimeters for linear axes. By moving all joints to the specified values, the robot moves to an unambiguous position and orientation in space.

### 3.1.2. Cartesian Robot Locations

A Cartesian location specifies the coordinates of a position in space using X, Y, and Z coordinates, and an orientation of the robot tool using yaw, pitch, and roll angles. For a more detailed discussion, see the Precise Documentation Library. Depending on the robot kinematics, there may be more than one set of joint angles that puts the robot's gripper at the same Cartesian location, so the Config property specifies additional information to determine an unambiguous set of joint angles.

### 3.1.3. Rail Position

If your robot includes an optional linear rail, the rail position is stored with each station.

### 3.1.4. Pallets

A pallet is a 1, 2, or 3-dimensional array of regularly spaced Cartesian Robot Locations. Once a pallet is defined, the pallet indices determines which column, row, and layer of the pallet is accessed. Many of the robot motion commands support pallets. Information about pallet handling is found in the document *PARobot Plug-In for TCS*.

## 3.2. Profiles

Profiles are descriptions of how a robot should move to a location. From within TCS, by default, you can access up to 20 profiles, numbered from 1 to 20. This limit can be easily increased for your application. These profiles are referenced by the *profile index*, from 1 to n, that is specified as a parameter to many commands. A profile describes the speed, acceleration, and motion path (e.g. straight-line or joint-interpolated) to be used.

### 3.2.1. Joint-Based Motion

If a profile Straight property is False, the robot moves each axis from its start to end position. The start and end times for all axes are coordinated, but the path the robot takes depends on the robot geometry. Most likely, the robot tool will not move in a straight line.

### 3.2.2. Straight-Line Motion

If a profile Straight property is set to True, the robot moves its tool in a straight line from the start to the end location. Depending on your robot, you may not be able to move between certain locations in a straight line.

## 3.3. Motion Considerations

### 3.3.1. Motion Timing

Most TCS motion-related commands do not wait for a motion to be complete. You can think of a *move* command as meaning "start moving to the specified location". The command reply is sent immediately after the robot starts moving. Subsequent TCS commands may be sent and processed while the robot is moving.

If a second motion command is sent while the referenced robot is moving, the second command is blocked and will not reply until the first motion is complete and the robot begins moving toward the second location.

If the second motion command is not sent until the robot is near the end of the first motion, the robot will decelerate to a stop and will begin accelerating for the 2nd move when the command completes. If you want a continuous blended move from one location to the next location along a path, it is important to issue commands quickly enough so that the robot does not stop. In many cases, blended motions can reduce the robot's cycle time by 10%-30% and will result in smoother motions.

If you want to wait for the robot to stop moving, you either issue a *waitForEom* command, or repeatedly issue a *state* command until the state is "idle".

Example: Use instruction blocking to synchronize signals with motion. Assume you are moving through locations 1, 2, 3 and 4, and you want a signal to be on during the move from 2 to 3. You can issue the following commands:

```
Move 1 5     'Starts moving to location 1 with profile 5
Move 2 5     'Waits for current then starts moving to location 2
Sig 13 1     'Turns on output 13
Move 3 5     'Waits for current then starts moving to location 3
```

```
Sig 13 0    'Turns off output 13
Move 4 5    'Waits for current then starts moving to location 4
WaitForEom  'Waits until the robot tops at location 4
```

## 3.3.2. Single Motions

A single robot motion consists of several parts. The motion profile specifies the parameters for each part.

1. Accelerate until the desired speed is reached.
2. Move at a constant speed.
3. Decelerate when near the end location.
4. Wait until the robot tool is at the end location, within a tolerance.

## 3.3.3. Blended Motions

Multiple motion commands are strung together to create a continuous motion provided that the command are received quickly enough. Motion profiles specify the parameters for each motion segment. Assume we are moving through locations 1 to location 2 using profiles 1 and 2.

1. Accelerate to the desired speed for using profile 1.
2. Move at the speed from profile 1 until the robot is near location 1.
3. Overlap the deceleration of profile 1 with the acceleration of profile 2 to transition to the move toward location 2.
4. Move at the speed from profile 2 until the robot is near location 2.
5. Decelerate when near location 2.
6. Wait until the robot is at location 2 based on the position error tolerance specified by profile 2.

# 4. Standard Commands

## 4.1. Command-Reply Syntax

Each command and reply is sent as an ASCII string. The format of the string depends on the mode settings:

TCP vs. Serial
PC vs. Verbose

| Modes | Command Syntax | Reply Syntax |
|---|---|---|
| TCP, PC | `command arg1 … argn LF` | `replycode data CRLF` |
| TCP, verbose | `command arg1 … argn LF` | `reply data CRLF` |
| Serial, PC | `robot command arg1 … argn LF` | `robot replycode data CRLF` |
| Serial, verbose | `robot command arg1 … argn LF` | `robot reply data CRLF` |

| | |
|---|---|
| `command` | is a command string such as "version" or "move". The command name must be spelled out completely. Abbreviations are not allowed. However the case of the command characters is ignored. |
| `arg1 … argn` | is a group of one or more numeric values or keywords that depend on the specific command. The arguments are separated by single spaces. |
| `replycode` | is a number that indicates the success or failure of the command. A value of 0 indicates success, a negative value is an error code. If an error code is returned, it is normally followed by a string that describes the error. |
| `reply` | is a string message or negative error code in the case of an error. |
| `data` | is the data that depends on the specific command |
| `robot` | is a number that indicates the destination of a command or the source of a reply. 0 indicates the status thread. Values > 0 indicate a robot. For a few errors, the robot number may be omitted, so the client should interpret a negative value as the `replycode` value. |
| `LF` | is an ASCII line-feed character (decimal value 10). The LF character may optionally be preceded by an ASCII carriage return (CR) character (decimal value 13), which is ignored. |
| `CRLF` | is an ASCII carriage return character (decimal value 13) followed by a line-feed character (decimal value 10). |

### 4.2. Verbose Mode

Verbose mode is meant to be used when interacting with a human client. For machine interfaces, PC mode is recommended. Serial verbose mode also allows the option of having the server echo characters.

### 4.3. Serial Communications

When using serial communications, all commands and replies occur on a single communications line. The messages to the status thread and robots are multiplexed on that line.

For commands, the first value indicates the destination of the command. A 0 means the status thread, and a value $n > 0$ indicates robot $n$. Each reply also contains a leading numeric value indicating its source.

The commands and replies for each destination do not block each other, so if you are using multiple threads, your client must be able to handle out-of-order messages. For example, if robot 1 is making a long move, and robot 2 is making a shorter move, you could see the following messages in PC mode.

```
Command:  1 move 1 5
Reply:    1 0
Command:  2 move 1 5
Reply:    2 0
Command:  1 waitForEom
Command:  2 waitForEom
Reply:    2 0
Reply:    1 0
```

For PC mode and non-echo Verbose mode, responses to commands are always atomic messages, so you can use the first field of the message to determine the source. But for Verbose mode with echo, responses from threads may be interspersed with the echoed characters so that the responses cannot be parsed.

### 4.4. Command Details

All the PC mode responses begin with a "0" if the command was successful, or a negative error code number. The response information in the tables below is appended after that response.

#### 4.4.1. General Commands

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| attach | arg1: Optional. If omitted, returns the attachment state.<br>0 = release the robot.<br>1 = attach the robot. | Attaches or releases robot. The robot must be attached to allow motion commands.<br>If arg1 is omitted, returns 0 if robot is not attached, -1 if the robot is attached. |
| base | arg1-4: Optional. If omitted, returns the current base value.<br>If specified,<br>arg1: base X offset<br>arg2: base Y offset<br>arg3: base Z offset<br>arg4: base Z rotation | Sets or gets the robot base offset. The robot must be attached to set the base. Setting the base pauses any robot motion in progress. If args omitted, response is X offset Y offset Z offset Z rotation |
| exit | none | Closes the communications link immediately. Does not affect any robots that may be active. |
| home | none | Homes robot associated with this thread.<br>Requires power to be enabled.<br>Requires robot to be attached.<br>Waits until the homing is complete. |
| homeAll | none | Homes all robots.<br>Requires power to be enabled.<br>Requires that robots not be attached. |
| hp | arg1: Optional. If omitted, returns the current power state.<br>0 = disable robot power<br>1 = enable robot power.<br>arg2: Optional.<br>0 or omitted = do not wait for power to come on.<br>> 0 = wait this many seconds for power to come on.<br>-1 = wait indefinitely for power to come on. | Enables or disables the robot high power. Returns an error if power does not come on within the specified timeout. |

| **Command**<br>Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| mode | arg1: 0 = Select PC mode,<br>1 = Select verbose mode.<br>If omitted, returns the current mode. | Sets or gets the response mode.<br>Telnet response is "1 Telnet_mode",<br>PC mode response is "0".<br>When using serial communications, the mode change does not take effect until one additional command has been processed. |
| mspeed | arg1: Optional. If omitted returns the global system (monitor) speed. Otherwise it must be a value between 1 and 100, which defines the new monitor speed as a percentage. 100 means full speed. | Gets or sets the global system speed<br>See documentation for DataID 601. |
| nop | none | Does nothing except return the standard reply. Can be used to see if the link is active or to check for exceptions. |
| payload | arg1: Optional. If omitted, returns the payload percent value for the current robot. Otherwise must be a value from 0 to 100 indicating the percent of the maximum payload the robot is carrying. | Gets or sets the payload percent of maximum for the currently selected or attached robot. If the robot is moving, waits for the robot to stop before setting a value. |
| pc | Accepts 2 or 5 arguments.<br>If 2 arguments:<br>arg1 = DataID of parameter.<br>arg2 = New parameter value.<br>If 5 arguments:<br>arg1 = DataID of parameter.<br>arg2 = Unit number, usually the robot number. (1 - N_ROB)<br>arg3 = Sub-unit, usually 0.<br>arg4 = Array index.<br>arg5 = New parameter value. | Parameter change. Changes a value in the controller's parameter database. If the New Parameter value is inside quote marks, a string value is written. Otherwise a numeric value is written. Updated values are not saved in flash unless a save-to-flash operation is performed (see DataID 901). |

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| pd | Accepts 1 to 4 arguments.<br>arg1 = DataID of parameter.<br>arg2 = Unit number, usually the robot number. (1-NROB)<br>arg3 = Sub-unit, usually 0.<br>arg4 = Array index. | Parameter display. Returns the value of a numeric parameter database item. See the Configuration and Parameter Database documentation.<br>Returns the numeric value of the specified database parameter. |
| reset | arg1 = The number of the robot thread to reset, from 1 to N_ROB. Must not be zero. | Resets the threads associated with the specified robot by stopping and restarting them. Any TCP/IP connections made by these threads are broken. This command can only be sent to the status thread. |
| selectRobot | arg1: Optional. If omitted returns the number of the currently selected robot.<br>If specified, indicates the new robot to be connected to this thread (1 to N_ROB) or 0 for none. | Changes the robot associated with this communications link. Does not affect the operation or attachment state of the robot. The status thread may select any robot or 0. Except for the status thread, a robot may only be selected by one thread at a time. |
| sig | arg1: The number of the digital signal to get or set.<br>arg2: Optional. If omitted, the signal value is returned. If specified, sets the signal to this value. | Gets or sets the specified digital input or output signal. If arg2 = 0, set the output to off. If arg2 = non-zero, set the output to on. |
| sysState | none | Returns the global system state code. Please see documentation for DataID 234. |
| tool | arg1-6: Optional. If omitted, returns the current tool transformation value. If specified, args are tool X, Y, Z yaw, pitch, roll. | Sets or gets the robot tool transformation. The robot must be attached to set the tool. Setting the tool pauses any robot motion in progress. If args omitted, response is tool: X Y Z yaw pitch roll |
| version | none | Returns the current version of TCS and any installed plug-ins. |

### 4.4.2. Location-Related Commands

See Location Class documentation for more details.

| **Command**<br>Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| loc | arg1: The station index, from 1 to N_LOC | Returns the specified location value. The value shown includes any offsets due to pallet indexing. For angles type, returns type code 1 followed by the station index followed by the angle values.<br>For Cartesian type, returns type code 0 followed by the station index followed by the values of X, Y, Z, yaw, pitch and roll. |
| locAngles | arg1: Specifies the station index.<br>arg2 - arg13: Optional. If any are specified, sets the location values. If none are specified, returns the angles values.<br>arg2 = axis 1 angle<br>arg3 = axis 2 angle, etc. | Gets or sets location angles. Error if you attempt to get angles from a Cartesian location.<br>If arg2 through arg13 are specified, sets new angle values. If more arguments than the number of axes for this robot, extra values are ignored. If less arguments than the number of axes for this robot, missing values are assumed to be zero.<br>Response is type code 1, followed by station index, optionally followed by axis angles. |
| locXyz | arg1: Specifies the station index.<br>arg2 - arg7: Optional. If specified, sets the location values. If none are specified, returns the Cartesian values.<br>arg2 through arg3 = X, Y, Z, yaw, pitch, roll values. | Gets or sets location Cartesian position. Error if you attempt to get Cartesian position from an angles type location.<br>If arg2 through arg7 are specified, sets new location values.<br>Response is type code 0, followed by station index, optionally followed by X, Y, Z, yaw, pitch roll. |
| locZClearance | arg1: Specifies the station index.<br>arg2: Optional. If omitted, returns the ZClearance property value. Otherwise specifies the new Zclearance property value. | Gets or Sets the Zclearance property for the specified location.<br>Response is station index followed by value. |

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| locConfig | arg1: Specifies the station index.<br>arg2: Optional. If omitted, returns the Config property value. Otherwise specifies the new Config property value. | Gets or Sets the Config property for the currently selected location.<br>Response is station index followed by value. |
| DestC | arg1: Optional. If omitted, assume value 0. Selects return value. | If arg1 = 1 or robot is moving, return the target Cartesian location of the previous or current move. If arg1 omitted or 0, and the robot is not moving, return the current Cartesian location. Response is: X Y Z yaw pitch roll config |
| DestJ | arg1: Optional. If omitted, assume value 0. Selects return value. | If arg1 = 1 or robot is moving, return the target Joint location of the previous or current move. If arg1 omitted or 0, and the robot is not moving, return the current Joint location. Response is: axis1 axis2 … axisn |
| HereJ | arg1: Specifies the station index. | Records the current position of the selected robot into the specified Location. The Location is automatically set to type "angles". |
| HereC | arg1: Specifies the station index. | Records the current position of the selected robot into the specified Location object. The Location object is automatically set to type "Cartesian". Can be used to change the pallet origin (index 1,1,1) value. |
| where | none | Returns the current position of the selected robot in both Cartesian and joints format. Response is: X Y Z yaw pitch roll axis1 axis2 … axisn.<br>In Telnet mode, the response is on two lines. |
| wherec | none | Returns the current Cartesian position and configuration of the selected robot. Response is: X Y Z yaw pitch roll config |
| wherej | none | Returns the current Joint position for the selected robot. Response is: axis1 axis2 … axisn |

### 4.4.3. Profile-Related Commands

See Profile Class documentation for more details.

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| Speed | arg1: Specifies the profile index.<br>arg2: Optional. If omitted returns the speed from the profile. If specified, sets the new speed as a percentage. 100 = full speed. | Gets or sets the speed property of the specified profile. The speed may be set to > 100 depending on your system configuration. |
| Speed2 | arg1: Specifies the profile index.<br>arg2: Optional. If omitted returns the speed2 from the profile.<br>If specified, sets the new speed2 as a percentage. 100 = full speed. | Gets or sets the speed2 property of the specified profile. Used for Cartesian moves. Normally set to 0. See the Profile Class documentation for details. |
| Accel | arg1: Specifies the profile index.<br>arg2: Optional. If omitted returns the accel from the profile.<br>If specified, sets the new accel as a percentage. 100 = maximum accel. | Gets or sets the Accel property of the specified profile. The maximum accel value depends on your system configuration. |
| AccRamp | arg1: Specifies the profile index.<br>arg2: Optional. If omitted returns the accel ramp from the selected profile.<br>If specified, sets the new accel ramp as a time in seconds. | Gets or sets the AccelRamp property of the specified profile. |

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| Decel | arg1: Specifies the profile index.<br>arg2: Optional. If omitted returns the decel from the selected profile.<br>If specified, sets the new decel as a percentage. 100 = maximum decel. | Gets or sets the Decel property of the specified profile. The maximum decel value depends on your system configuration. |
| DecRamp | arg1: Specifies the profile index.<br>arg2: Optional. If omitted returns the decel ramp from the selected profile.<br>If specified, sets the new decel ramp as a time in seconds. | Gets or sets the DecelRamp property of the specified profile. |
| InRange | arg1: Optional. If omitted returns the InRange property from the selected profile.<br>If specified, sets the new InRange from -1 to 100. | Gets or sets the InRange property of the selected profile. -1 means do not stop at the end of motion if blending is possible. 0 means always stop but do not check the end point error. > 0 means wait until close to the end point. Larger numbers mean less position error is allowed. |
| Straight | arg1: Specifies the profile index.<br>arg2: Optional. If omitted returns the Straight property from the selected profile.<br>If specified, -1 means follow a straight-line path. 0 means follow a joint-based path. | Gets or sets the Straight property of the selected profile. If 0, the robot axes move in a coordinated manner with nominally constant velocity. The path the robot tool takes depends on the robot geometry. |

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| Profile | arg1: Specifies the profile index.<br>arg2 - arg9: Optional. Must all be omitted or specified together. If specified, contain the new profile property data.<br>arg2: Speed<br>arg3: Speed2<br>arg4: Accel<br>arg5: Decel<br>arg6: AccelRamp<br>arg7: DecelRamp<br>arg8: InRange<br>arg9: Straight | Get or set all profile properties at once. Get returns the properties as follows: Speed Speed2 Accel Decel AccelRamp DecelRamp InRange Straight. |

## 4.4.4. Motion Related Commands

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| halt | none | Stops the current robot immediately but leaves power on. |
| Move | arg1: The index of the location to which the robot moves.<br>arg2: The profile index for this move. | Moves to the location specified by the station index using the specified profile. Requires that the robot be attached. |
| moveAppro | arg1: The index of the location to which the robot moves.<br>arg2: The profile index for this move. | Approaches the location specified by the station index using the specified profile.<br>Requires that the robot be attached.<br>Similar to "move" except that the Zclearance value is included.<br>Requires that the robot be attached. |
| moveExtraAxis | arg1: The destination position for the 1st extra axis.<br>arg2: Optional. The destination position for the 2nd extra axis, if any. | Posts a move for one or two extra axes during the next Cartesian motion. Does not cause the robot to move at this time. Only some kinematic modules support extra axes. Requires that the robot be attached. |

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| MoveOneAxis | arg1: The number of the axis to move.<br>arg2: The destination position for this axis.<br>arg3: The index of the profile to use during this motion. | Moves a single axis to the position specified arg2 using the profile determined by arg3. Requires that the robot be attached. |
| MoveC | arg1: The profile index to use for this motion.<br>arg2 .. arg7: The Cartesian coordinates specifying where the robot should move:<br>X Y Z yaw pitch roll.<br>arg8: Optional. If specified, sets the Config property for the location. | Moves the robot to the Cartesian location specified by the arguments. Requires that the robot be attached. |
| MoveJ | arg1: The profile index to use for this motion.<br>arg2 … argn: The joint angles specifying where the robot should move. The number of arguments must match the number of axes in the robot. | Moves the robot to the angles location specified by the arguments. Requires that the robot be attached. |
| releaseBrake | arg1: The number of the axis whose brake should be released. | Releases the axis brake. Overrides the normal operation of the brake. It is important that the brake not be set while a motion is being performed. This feature is used to lock an axis to prevent motion or jitter. |
| setBrake | arg1: The number of the axis whose brake should be set. | Sets the axis brake. Overrides the normal operation of the brake. It is important not to set a brake on an axis that is moving as it may damage the brake or damage the motor. |
| state | none | Returns state of motion. This value indicates the state of the currently executing or last completed robot motion. For additional information, please see 'Robot.TrajState' in the GPL reference manual. |

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| waitForEom | none | Waits for the robot to reach the end of the current motion or until it is stopped by some other means. Does not reply until the robot has stopped. |
| zeroTorque | arg1: Zero to disable torque mode for the entire robot. One to enable torque mode for the bits specified by arg2.<br>arg2: The bit mask specifying the axes to be placed in torque mode if arg1 is 1. The mask is computed by OR'ing the axis bits:<br>1 = axis 1<br>2 = axis 2<br>4 = axis 3<br>8 = axis 4, etc. | Sets or clears zero torque mode for the selected robot. Individual axes may be placed into zero torque mode while the remaining axes are servoing. |

### 4.4.5. Vision-Related Commands

See the PreciseVision documentation for more information

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| vprocess | arg1: The name of the vision process | Executes vision process. Sends a vision process request to PreciseVision. The process names are shown in the PreciseVision Process Manger window. If successful, returns the vision result count. |
| vresultInfo | arg1: Optional. Specifies the name of the vision result to return. If omitted, the default result is returned.<br>arg2: Optional. Specifies the array index of the named vision result.<br>arg3: Optional. Specifies the info array index. If omitted, all info results are returned. | Returns values from the vision result info array. The result is from the last vision process executed.<br>Response is one or more info values separated by spaces. |

| **Command** Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| vresultInspect | arg1: Optional. Specifies the name of the vision result to return. If omitted, the default result is returned. arg2: Optional. Specifies the array index of the named vision result. | Returns inspection information from the vision result. Response is the InspectActual property value followed by the InspectPassed property value. The meanings of these properties depend on the vision tool executed. |
| vresultLoc | arg1: Optional. Specifies the name of the vision result to use. If omitted, the default result is used. arg2: Optional. Specifies the array index of the named vision result. | Returns the Cartesian location information from the vision result. The result is from the last vision process executed. Response is: X Y Z yaw pitch roll |

## 5. System Reference

### 5.1. Project Files

The standard Tcp_cmd_server project consists of the files shown below. Of these files, only Startup.gpl should normally be modified since it configures standard TCS. All other customizations should be put in plug-in modules.

5.1.1. **Startup.gpl** - Global entry point into project (launches 1 thread for each robot).

Defines the number of robots supported by TCS. The constants near the start of this file specify the TCP ports used and the number of robots supported (from 1 to 8). To support more robots you must modify *StartMain* and define additional top-level routines in this file.

5.1.2. **Main.gpl** - Contains the main loops for communications and robot command handling. All TCP I/O is performed in this file.

5.1.3. **Class_command.gpl** - Defines the Command and RobData classes which hold all communications and robot information within TCS. You can add fields to these classes if necessary, but it is recommended that any plug-in data be kept within the plug-in module. The command parser and top-level handler is a method of the Command class.

5.1.4. **Class_gpofile.gpl** - Defines the GpoVar and Gpofile classes. Used by loadfile and storefile to parse and write .gpo files.

5.1.5. **Class_Station.gpl** - Defines the Station class that holds all robot location, rail, and pallet offset data.

5.1.6. **Class_StringList.gpl** - Defines the StringList class used for passing strings between threads for serial communications.

5.1.7. **Class_vector3.gpl** - Defines the Vector3 class used for manipulating 3-dimensional vectors while defining pallets.

5.1.8. **Cmd.gpl** - Contains the handlers for all standard commands.

5.1.9. **Custom.gpl** - Contains a template for a custom plug-in module. Use this as an example to create custom commands.

5.1.10. **Globals.gpl** - Defines and initializes global variables and constants used by the standard TCS commands. The constants N_PROF and N_LOC may be modified to increase the number of profiles or locations available for commands.

5.1.11. **Functions.gpl** - Contains global functions used by the handler and various commands. May be useful in custom commands.

5.1.12. **Load_Save.gpl** - Contains commands for loading and storing station and profile information from .gpo files.

5.1.13. **Projects.gpr** - A standard GPL project file for the this project.

## 5.2. Threads

TCS uses multiple independent threads (or tasks). There is one thread allocated for status and two threads for each robot. This arrangements allows commands for status and robots to operate independently and not block each other.

| **Thread Name** | **Description** |
|---|---|
| Tcp_cmd_server | Main thread that starts all other threads. Handles TCP or serial status communications and commands if status is enabled. Otherwise exits after startup. |
| SerialCom*n* | Serial communications for robot *n*, *n* = 1 to 8 |
| SerialReceiver | Receives all serial messages and dispatches them to the appropriate SerialCom*n* thread. |
| SerialSender | Sends all serial messages. |
| TcpCom*n* | Handles TCP communications for robot *n*, *n* = 1 to 8 |
| Rob*n* | Command handler for robot *n, n* = 1 to 8 |

For each robot, the communications thread reads commands from the client over TCP and sends replies back to that client. When a complete command is received, it is forwarded to the associated robot thread which executes the command. The reply is forwarded back to the communications thread for transmission to the client.

If enabled, the status thread cannot handle robot-related commands. This thread handles both communications and command execution.

## 5.3. RobData Class

All robot information is represented by objects of the RobData Class. There is a single object of this class for each robot being controlled by TCS. This object contains all the location and profile information for that robot.

5.3.1. RobData Class Fields - The RobData class object contains the following fields:

| **Command Object Field** | **Description** |
|---|---|
| RobNum | The number of the robot associated with this command, or 0 if no robot (status thread). |
| RobNumAxes | The number of axes on this robot, not including any optional gripper. |
| RobType | The type of this robot from DataID 2007. |
| GripAxis | The axis number of the servoed gripper, or 0 if no servoed gripper. |
| RailAxis | The axis number of a linear rail, or 0 if no linear rail. |
| ThetaAxis | The axis number of a theta axis, or 0 if no theta axis. |
| InactiveGripAxis | The axis number of the second (inactive) gripper, or 0 if no dual gripper installed. |
| ActiveGripper | The Active gripper,<br>GripperA = 0,<br>GripperB = 1,<br>0 If Not dual gripper. |
| RobProf() | An array of profiles for this robot used by commands such as Move. |
| RobSta() | An array of station objects for this robot used by commands such as Move. |

## 5.4. Station Class

The station class holds location and additional information used by the standard commands and the enhanced commands found in the PARobot plug-in. The fields in this class are described in the table below:

| **Station Object Field** | **Description** |
|---|---|
| flags | Bit flags that determine the station type.<br>STA_FL_PAL = &H0001 set means pallet-relative. Clear means no reference frame used.<br>STA_FL_VERT = &H0010 set means vertical access. Clear means horizontal access. |
| loc | The location value. May be relative to the reference frame if the type is pallet-relative. |
| rail | The rail position for this station. |
| z_above | The offset in the Z direction for horizontal access stations. |
| z_grasp_offset | The approach offset in the Z direction when a part is being grasped. Used by enhanced instructions. |
| frame | The reference frame that contains pallet information. |

## 5.5. Command Class

All received messages are represented by objects of the Command class. Each communications thread creates a single object of this class. All information about the command is contained within this object.

Each Command object is associated with a single RobData object. This association may change based on use of the SelectRobot command.

5.5.1. Command Class Fields - The command class object contains the following fields:

| **Command Object Field** | **Description** |
|---|---|
| InMsg | A string that contains the raw input from the most recent command. |
| Reply | A string that will receive the reply data from the command handler. |
| Nparm | After parsing. The number of arguments passed to this command |
| sData() | After parsing. A string array containing the arguments for this command.<br>sData(0) contains the actual command.<br>sData(1) contains argument 1, etc. |
| cData() | After parsing and calling StringToDouble(). A Double array containing the numeric values of the corresponding elements of sData().<br>cData(1) contains argument 1, etc |
| IsStatus | True if this is a status thread with no robot connected. |
| NewRobNum | A mailbox for requesting a robot number used by command SelectRobot. |
| Rob | A RobData class object for the robot associated with the current communications thread. |

5.5.2. Command Object Use

The command object is used as follows:

1. At startup, each communications thread creates a single Command object and RobData object.
2. For serial communications, a single thread receives all messages and dispatches them to the appropriate communications thread. For TCP, each communications thread receives its own messages.
3. When a message is received, the raw data is placed in the Command object InMsg field.
4. For robots, the object is passed to the robot thread for handling. For status, the object is passed directly to the top-level handler in the local thread.

5. The top-level handler parses the data in InMsg and fills in Nparm and sData(). It sets a default Reply value. It then calls the specific command handler for this command, passing the object.
6. The specific command handler optionally calls StringToDouble() to fill in cData(). It the uses the object fields to execute the command. It may modify the location or profile data in the object.
7. The specific command handler modified the reply string if necessary, and returns to the top-level handler.
8. The top-level handler returns the object to the robot thread or status thread.
9. The robot thread returns the object to the corresponding communications thread.
10. The communications or status thread transmits the reply back to the client over the communications link. For serial communications, all message are sent by a single thread.

## 5.6. Command Handlers

Each command to TCS is handled by a separate command handler routine. The standard commands are found in the file Cmd.gpl. GPL Delegates are used to call these handlers.

5.6.1. Command Handler Names - The name of a command handler is the command name, with the prefix "Cmd_" pre-pended to the start. For example, the command "move" is handled by a procedure named "Cmd_Move". Since the command is part of a procedure name, it must follow GPL naming conventions (no embedded spaces or special characters other than "_"). And the case of the command and the handler is ignored.

5.6.2. Parameters - The command handler is passed two arguments

Parameter 1 - A Command class object which contains all inputs to the command. The command may reference the command name, arguments, robot number etc through this object.

Parameter 2 - A string variable passed by reference that receives any special data to be returned as the reply to this command. By default, the reply is set to "0" in PC mode and "Ok" if in Telnet mode, so there is no need to change it if these values are appropriate.

5.6.3. StringToDouble() - Since many command reference numeric arguments, this method is a convenient way to convert the command arguments to Double values and store them in the cData() field.

5.6.4. Telnet vs. PC mode - Many times the reply value is different depending on the mode. The global variable *bTelnetMode* is True in Telnet mode or False if in PC mode. This variable may be tested to modify the reply contents.

## 5.7. Initialization

TCS makes use of the module initialization procedure *Init* which is automatically called for each module found during project startup. The *Init* procedures are called in the order in which the modules are found in the source files and the source files are found in the project.

## 5.8. Exceptions

If an exception occurs during execution of a command handler or if the handler explicitly throws an exception, the exception is trapped by the top-level command handler which sends the standard reply error code and message string instead of the normal command reply. Individual command handlers may trap exceptions and handle them directly.

If a robot-related exception occurs asynchronously between commands, when the next command is received, it is not processed. Instead, the exception error code is returned as the reply to that command.

If an exception occurs within the communications thread, a system message is logged and the connection is closed. You can check the system message log from the web-based operator control panel or from GDE.

## 5.9. Hook Procedures

Plug-in modules use procedures called "hooks" to perform custom operations. A hook procedure is called from within some standard procedure to augment the standard operation.

5.9.1. Command.AddPlugin -- This method is called by a plug-in to notify TCS that a plug-in has been installed. Normally this call is made from the plug-in "Init" routine. A sample call is shown below:

```
Command.AddPlugIn(Version, "MyModule")
```

where: `Version` is the version string defined in the top-level and `"MyModule"` is the name of the module that contains the plug-in.

TCS can now use the Command.RunPluginHook method to call routines that are contained in the plug-in module.

5.9.2. cmd.RunPluginHook -- This method is called from within TCS routines to call a custom routine within the plug-in.

5.9.3. Standard Hook Procedures -- The following hook procedures are used by TCS.

Hook_InitCommand -- This procedure is called whenever a new command object is created. It can be used to initialize data within the command object or to initialize global data that depends on the robot number.

Hook_LoadFile -- This procedure is called by the LoadFile command to handle custom data in a .gpo file when loading a file.

Hook_StoreFile -- This procedure is called by the StoreFile command to handle custom data in a .gpo file when storing a file.

## 6. Customization

Because TCS is open-source, you can change anything you want. However, it is intended that you change only a few items in the standard core programs and make more extensive changes in the form of plug-ins.

### 6.1. Core Program Customization

The following table summarizes the variables in the core TCS files that you might want to modify.

| **Variable** | **File** | **Description** |
|---|---|---|
| NumRobots | Startup.gpl | The number of robots supported by this package. Default is 1. Can be set from 1 to 8. If > 8 is desired, you will need to create additional Com*n* and Rob*n* Sub procedures. |
| NumStatusPorts | Startup.gpl | The number of status ports supported. Default is 1. Can be set from 0 to 4. |
| CommType | Startup.gpl | A code indicating the type of communications to be used:<br>0 = TCP/IP communications over Ethernet.<br>1 = RS-232 communications.<br>2 = RS-232 communications with echoing of input. |
| SerialPort | Startup.gpl | A string containing the device for serial communications. Ignored in TCP/IP mode. |
| Status*n*Port | Startup.gpl | The TCP port used by status thread *n*. Default is 10000 + *n* - 1. Ignored in serial mode. |
| Robot*n*Port | Startup.gpl | The TCP port used by robot *n*. Default is 100*n*00. *n* = 1 to 8. Ignored in serial mode. |
| Version | Globals.gpl | The string displayed by the Version command. Modify this if you make significant changes to TCS. |
| N_LOC | Globals.gpl | The number of locations that can be defined. Default is 20 |
| N_PROF | Globals.gpl | The number of motion profiles that can be defined. Default is 20. |

### 6.2. Plug-In Creation

Major enhancements to TCS should be in the form of a plug-in. A plug-in is a GPL program file that contains one or more modules or classes that contain custom routines that may be called to handle commands from the TCS client. See the *System Reference* section for details on some of the topics mentioned in this section.

The standard file Custom.gpl contains a template module and command handler. You can copy this file and edit it to create your own plug-in. Then use GDE to add the plug-in file to your TCS project. The commands in the plug-in are automatically available for use.

## 6.3. Example: Custom.gpl

This section describes in detail the parts of the custom plug-in file found in Custom.gpl.

6.3.1. Top-Level Module - The top-level module name must be unique among all the TCS modules. This name normally should reflect the name of your plug-in.

At the top level of this module, you should create any global definitions, variables or data structures.

You should define a Const String named Version to hold the name and version information for your plug-in.

6.3.2. Module Init Procedure - A procedure named *Init* should be defined to perform special plug-in initialization when your TCS project starts. This routine is automatically called by GPL.

This procedure should inform TCS of the new plug-in by calling the *Command.AddPlugin* method.

This call allows TCS to show your version information in the Version command and also enables hook routines to be called.

6.3.3. Hook_InitCommand Procedure - The procedure named "MyModule.Hook_InitCommand" is called whenever a new command object is created. It can be used to initialize data within the command object or to initialize global data that depends on the robot number. Normally a command object is only created for the status thread and for each robot. This routine is called from the communications thread, so it should not attempt to access a robot. If you do not need to initialize data, you do not need to define this procedure.

6.3.4. Command Handler Procedure - A command handler procedure is called whenever a command is received from the client. There is a separate procedure for each command. The procedure name determines the command name. Each handler has the name Cmd_*name* where *name* is the command name. Once the handler is in place, TCS automatically routes the matching command to the handler.

6.3.5. Command Handler Parameters - The first command handler parameter is a Command class object that contains the parsed arguments from the command.

See the *Command Class* section above.  In this example the name *Cmd* is used as the Command class object parameter.
To access the arguments as numeric (Double) values, you can call the method Cmd.StringToDouble(0) and then access Cmd.cData().  Or you can convert the Cmd.sData() values yourself.

The second command handler parameter is a ByRef string variable that contains the reply to be sent upon completion of this command.  In this example the name *Reply* is used as the reply string.  Upon entry, Reply is set to "0" if TCS is in PC mode and "Ok" if TCS is in Telnet mode.  You can change the Reply string to any value you wish and that string will be sent back to the client.  We recommend you continue to follow the reply convention described in the section *Quick Start > Commands* above.

## A. <u>Appendix - Load Save Plug-in</u>

This plug-in supports commands to load and save the station location and profile data associated with TCS commands. It is found in the file *Load_save.gpl* and is included by default in the standard distribution, so no special action is required to access it.

This plug-in allows you to load TCS data from a .gpo file, and to store modified data to a .gpo file. By default the file *Tcs.gpo* (which is included in the standard distribution) is used. Inside this file, the stations and profiles for robot 1 are saved. If you wish to store data for additional robots, you will need to use the Precise Guidance Development Environment (GDE) or a text editor to add location and profiles to the file. You must use GDE version 3.1.0.3 or later or the file will be corrupted.

Because the GDE editor does not support user-defined objects, the data is stored in arrays that are used to initialize the station objects used by TCS.

The following table shows the use of the tcs.gpo variables for robot 1. For additional robots, the suffix "_n" is used instead of "_1", where "n" is the robot number.

| **File Variable** | **Station Field** |
|---|---|
| tcs_sta_flags_1 | flags |
| tcs_sta_loc_1 | loc |
| tcs_sta_rail_1 | rail |
| tcs_sta_z_above_1 | z_above |
| tcs_sta_z_grasp_offset_1 | z_grasp_offset |
| tcs_sta_frame_1 | frame |
| tcs_prof_1 | N/A |

Note that for all these arrays, element 0 is present but not used.

The following TCS commands support loading and storing the data:

| **Command**<br>Case ignored | **Arguments** | **Description, responses are for PC mode. Telnet mode is similar but more descriptive.** |
|---|---|---|
| LoadFile | arg1: Optional. The name of the file to load. If omitted, the file *Tcs.gpo* in the current project folder is assumed. | Loads all location and profile data from the specified file. All previous location and profile data is lost.<br><br>By default, the file is loaded from the current TCS project folder. However if the file name begins with "/" it may be located anywhere. For example "/flash/data/myfile.gpo".<br><br>Only data defined in the Station.LoadFile method or a "Hook_LoadFile" plug-in is loaded. |
| StoreFile | arg1: Optional. The name of the file to write. If omitted, the file *Tcs.gpo* in the current project folder is assumed. | Writes the station and profile data for the current robot to the specified file.<br><br>By default, the file is written to the current TCS project folder. However if the file name begins with "/" it may be located anywhere. For example "/flash/data/myfile.gpo".<br><br>Any previous data in the file is lost. If you want to update a file, you must first load the file, change the data, then store it again.<br><br>Only data defined in the Station.StoreFile method or a "Hook_StoreFile" plug-in is stored. |

## B. Appendix - Error Codes

TCS uses standard GPL error codes in the command responses. They are described in the Precise Documentation Library. In addition a number of custom error codes are defined in standard TCS, in the file *Globals.gpl*. They are shown below.

| **Symbol** | **Number** | **Error message text** |
|---|---|---|
| EcPmm | -2800 | *Warning Parameter Mismatch* |
| EcNoParm | -2801 | *Warning No Parameters |
| EcIllMove | -2802 | *Warning Illegal move command* |
| EcInvJa | -2803 | *Warning Invalid joint angles* |
| EcInvCc | -2804 | *Warning: Invalid Cartesian coordinate values* |
| EcUnknown | -2805 | *Unknown command* |
| EcEx | -2806 | *Command Exception* |
| EcIO | -2807 | *Warning cannot set Input states* |
| EcCmdSrv | -2808 | *Not allowed by this thread* |
| EcInvRobType | -2809 | *Invalid robot type* |
| EcInvSerCmd | -2810 | *Invalid serial command* |
| EcInvRobNum | -2811 | *Invalid robot number* |
| EcBadRobSel | -2812 | *Robot already selected* |
| EcModNotInit | -2813 | *Module not initialized* |
| EcInvLocIdx | -2814 | *Invalid location index* |
| EcUndefLoc | -2816 | *Undefined location* |
| EcUndefProf | -2817 | *Undefined profile* |
| EcUndefPal | -2818 | *Undefined pallet* |
| EcNoPallet | -2819 | *Pallet not supported* |
| EcInvStaIdx | -2820 | *Invalid station index* |
| EcUndefSta | -2821 | *Undefined station* |
| EcNotPallet | -2822 | *Not a pallet* |
| EcNotOrigin | -2823 | *Not at pallet origin* |