TPanel

Reference guide

Written by Andreas Theofilu <andreas@theosys.at>

Table of contents

Abbildungsverzeichnis

Introduction

TPanel is an emulation of some AMX G4 touch panels. The panels used to reverse engineer the communication protocol and the behavior were a *AMX MVP-5200i* and a *AMX NXD-700Vi*.

This manual describes the commands implemented and some specials of this program. **TPanel** was designed for *NIX desktops (Linux, BSD, …) as well as Android operating systems version 10 or newer. Currently there exists no Windows version and there probably never will be one.

The software uses internally the Skia[1] library for drawing all objects and the Qt 5.15[2] library to display the objects. **TPanel** is written in C++. This makes it even on mobile platforms fast and reliable. It has the advantage to not drain the accumulator of any mobile device while running as fast as possible. Compared to commercial products the accumulator lasts up to 10 times as long.

Why does TPanel exist?

I'm a professional programmer and years ago I got in touch with AMX. I developed solutions for residential requirements in NetLinx with different AMX panels. With time the customers wanted to have the surface on their phone or a tablet and there was (is?) only one solution available on the market. While this software is very good and supports everything up to TP5 commands, the license is rather expensive. For me as a private person, too expensive. But I bought some used AMX controllers in the Internet and build my own smart home where I can control lights, hazard and my consumer electronics (TV, radio, audio, video, …). I bought also some used AMX panels but had no luck with the accumulators in them. To buy a new one from AMX was too expensive and it didn't pay off for such old devices. Buying a license for the commercial version (TPControl) was also no option because I would need 4 of them. So I decided to program my own surface and it should behave as a real device from AMX.

Picture 1: Main window with my surface

1 Skia is an open source 2D graphics library which provides common APIs that work across a variety of hardware and software platforms. It serves as the graphics engine for Google Chrome and Chrome OS, Android, Flutter, and many other products. Skia is sponsored and managed by Google, but is available for use by anyone under the BSD Free Software License. While engineering of the core components is done by the Skia development team, we consider contributions from any source.

2 Qt is a full development framework with tools designed to streamline the creation of applications and user interfaces for desktop, embedded, and mobile platforms.

Why AMX?

There are a lot of possibilities to build a smart home. Makers can use a Raspberry PI or an Arduino or something similar. You can create circuits to make serial ports, infrared control, I/O ports and relays. This is a lot of effort and you may spent up to several hundred Euros to build just a controller.
On the other side you can control your lights, the hazard and some modern TVs over Alexa and co. While this works it has the disadvantage to need a internet connection and there is a big cloud behind. If the internet is not available for whatever reason, you can't control anything.
For me some essential points are that I want to be absolutely independent of any internet connection and any cloud. I wanted to have a system which needs only a local network. Therefor I need a controller handling the smart home. It should be cheap and easy to handle. Since AMX offers all the necessary software to program without the need of a company account it is easy to get the knowledge to program a AMX controller with NetLinx. Beside this it is really easy to get a used AMX controller from eBay. I bought mine for about 50 Euros. Even some older equipment like volume controllers (AXB-VOL-3) are still available on eBay. With a *NI-3100* controller you can do everything you need to make your home smart. If you like, you can try to find some G4 panels also on the internet and you will find them.
To make it short: AMX is a fast and cheap way to implement just the software to make a smart home if you buy used equipment. In most cases you need no additional hardware (beside a local network). On the other side you must be interested in programming and you must be willing to learn an easy 3rd generation language like NetLinx is. This are the reasons for me to use AMX.

Old *discontinued* equipment

Device	Description
NI-700	The AMX NI-700 was designed to meet the needs of single room requirements while keeping cost in mind all in a 1RU box. The unit can control a limited number of video players, projectors, lights, thermostats, and other electronic equipment. The NI-700 is ideal for classrooms, conference rooms, hotel rooms and so much more. **Includes**: 2-pin 3.5 mm mini-Phoenix female PWR connector, 4-pin 3.5 mm mini-Phoenix female connector, 6-pin 3.5 mm mini-Phoenix female I/O connector, CC-NIRC IR Emitter
NI-900	The AMX NI-900 was created to automate and control numerous items in 1 large room or several small rooms. The NI-900 is ideal since it can support several different devices with numerous different communication formats. Common applications include hotel rooms, home theaters, and other environments. The NI-900 is configured to to control a small number of lights, thermostats, flat panels, and other audio video equipment. **Includes**: 2-pin 3.5 mm mini-Phoenix female PWR connector, 6-pin 3.5 mm mini-Phoenix female I/O connector, Three CC-NIRC IR Emitters, Two 4-pin 3.5 mm mini-Phoenix female connectors.

Device	Description
NI-2100	The AMX NI-2100 was designed for the automation and control of medium sized rooms and multiple room applications. The unit features 64MB of RAM and 3 configurable RS-232 / RS-422 / RS-485 serial ports. Programming the NI-2100 is simple since it is device discovery enabled offering several functions definitions for standardizing devices as well as default touch panel button assignments, and control and feedback methods. **Includes**: 2-pin 3.5 mm mini-Phoenix (female) PWR connector, 4-pin 3.5 mm mini-Phoenix (female) AxLink connector, 6-pin 3.5 mm mini-Phoenix female I/O connector, 8-pin 3.5 mm mini-Phoenix female Relay connector, Two CC-NIRC IR Emitters, Two removable rack ears.
NI-3100	The AMX-3100 was designed for large rooms or even multiple rooms where you need the ultimate control and automation. The controller can control numerous items including audio/video conferencing, projectors, DVD and Blu-Ray players, lights, thermostats and other electronic equipment found in larger rooms. Not only can it accomplish what you need now it can also provide solutions for future needs with its easy expansion capabilities. Installation is easy with device discovery enabled and performance is top notch with the speedy processor and 64MB of RAM. **Includes**: 2-pin 3.5 mm mini-Phoenix (female) PWR connector, 4-pin 3.5 mm mini-Phoenix (female) AxLink connector, 10-pin 3.5 mm mini-Phoenix (female) I/O connector, Two 8-pin 3.5 mm mini-Phoenix female Relay connectors, Two CC-NIRC IR Emitters, Two removable rack ears.
NI-4100	The NI-4100, part of the NI Series of Master Controllers, is geared to meet the high-end control and automation requirements of the most sophisticated and complex commercial and residential installations. This controller integrates the largest number of devices including DVD players, projectors, lighting, thermostats and other electronic equipment. In technology-intensive environments, this solution can be used to accommodate the future addition of more devices and control capabilities
AXB-VOL-3	The AXB-VOL3 Three-Channel Volume Control provides three audio volume control channels. Each line-level channel, opto-isolated from system ground, can be configured for balanced or unbalanced line operation. The AXB-VOL3 is programmable for 128 steps of audio level, audio mute, variable ramp speed and level presets. The AXB-VOL3 connects to NetLinx control systems using the 4-wire AXlink data/power bus; it can be used for remote or rack mount applications.

Device	Description
NXC-VOL4	NXC-VOL4 by AMX offers four discrete volume control channels with LED feedback and is programmable for mono or stereo operation, and balanced or unbalanced audio connections. Programmed features such as audio levels, audio mute, variable ramp speeds and preset levels. Use the on-board jumpers to set the gain/attenuation (Unity, Pro level (+4 dBu) to Consumer level (-10 dBu) conversion, or Consumer level to Pro level on each channel). NetLinx Control Cards provide flexible, modular building blocks for creating advanced control applications.
EXB-COM2	ICSLan Device Control Boxes allow users to manage devices remotely from a Controller over an Ethernet network. This provides a beautifully simple method for a centralized control environment allowing users to share a controller among multiple smaller rooms versus controllers in every room. Ethernet has become the industry standard for connecting devices and the ICSLan Device Control Boxes make it easy to introduce control to equipment such as projectors located extended distances from a Controller. Additionally, the number of ports on an AMX Controller can be expanded when all ports are fully populated. Because they employ Native NetLinx technology, it is extremely simple to add an EXB to an AMX installation.

Programming

Overview

You can program **TPanel** using the commands in this section, to perform a wide variety of operations using Send_Commands and variable text commands.
A device must first be defined in the *NetLinx* programming language with values for the Device: Port: System (in all programming examples - Panel is used in place of these values and represents **TPanel** program).

Touch Gesture Recognition

TPanel supports currently only one touch gesture to open the setup dialog. With a pinch gesture it is possible to open the setup dialog. It can be used on any device with a touch screen.

Setup Dialog

The setup dialog allows the setting of different things. It consists of *tabs* on the top allowing to select the wanted page. Currently 4 pages are available:

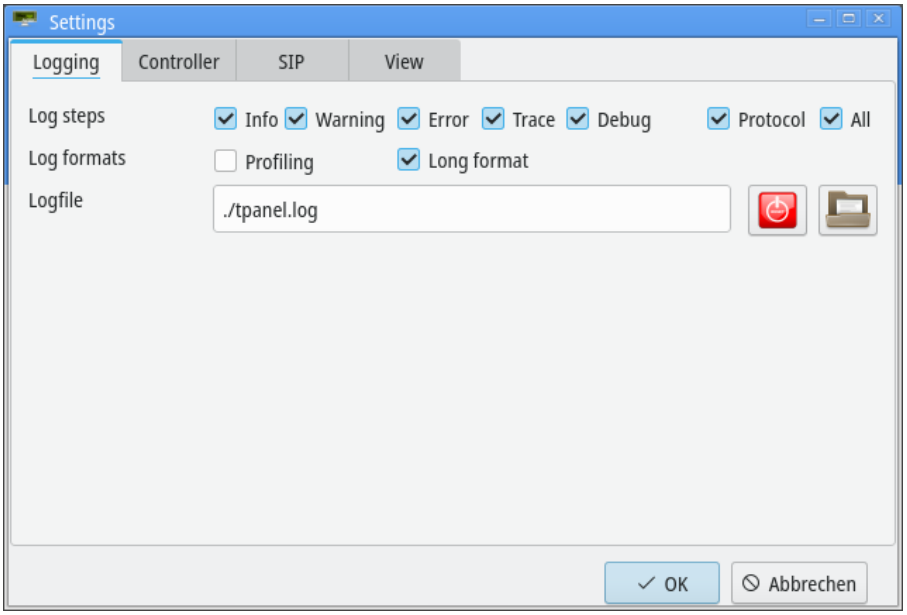
- Logging → Settings for the logfile.
- Controller → Everything about the controller.
- SIP → *Session Initiation Protocol* used for phone calls (currently not implemented)
- View → Some settings about visual effects.

Logging

The logging is meant to be enabled in case of problems. For example if you find a situation where the program crashes. In such a case logging can help the developers to find the course for the crash.

On mobile devices logging is disabled by default. The reasons are, that a special permission to the disc is necessary and by default the logfile is in a place where the user have no access to it. Enabling logging means also to put the logfile somewhere on the disc where a user has access to it. This can easily be done with a file dialog. Defining the path and name of the logfile also asks for permissions in case they are not already granted.

Attention!
Enabling the option `Trace` or all log levels on a mobile device will create a huge file in a short time. It is possible that your device becomes unusable!



Picture 1: Log settings

Log steps

The logging is based on states. This means that every stage of logging can be enabled or disabled independently of the other stages. There exists also two shortcuts: *Protocol* and *All*.

The *Protocol* stage is a combination of `Info`, `Warning` and `Error`.

The *All* stage enables all stages. This produces a lot of output and should not be used on a mobile device. When this is enabled everything is logged into a file. Because the program uses a lot of threads inside the content of the logfile may look funny sometimes. It is not possible to tell between the output of the threads.

If you're not a developer I would suggest to disable logging at all. Especially on a mobile device.

Log formats

Profiling

If this is enabled together with the *Trace* log option, a time stamp is printed at the end of each method.

```
TRC    75,     {entry TExpat::parse()
TRC    34,      {entry TValidateFile::isValidFile(const string& file)
TRC     ,       }exit TValidateFile::isValidFile(const string& file) Elapsed
time: 2508[ns] --> 0s 0ms
TRC     , Parsing XML file /usr/share/tpanel/map.xma
TRC     ,     }exit TExpat::parse() Elapsed time: 43929457[ns] --> 0s 43ms
TRC   318,     {entry TExpat::getElementIndex(const string& name, int* depth)
TRC     ,      }exit TExpat::getElementIndex(const string& name, int* depth)
Elapsed time: 4511[ns] --> 0s 0ms
```

The elapsed time is printed in nanoseconds as well as in seconds and milliseconds.

Long format

This adds additional information's like a time stamp. If *Trace* is enabled you'll see also the file name where the class is located along with the line number where the message was executed from.

```
2022-02-05 14:20:06 TRC   372, tsocket.cpp         ,        {entry TSocket::readAbsolut(char *buffer, size_t size)
2022-02-05 14:20:06 TRC   311, tsocket.cpp         ,         {entry TSocket::receive(char* buffer, size_t size)
2022-02-05 14:20:06 TRC     , tsocket.cpp         ,         }exit TSocket::receive(char* buffer, size_t size) Elapsed time: 3051[ns]
--> 0s 0ms
2022-02-05 14:20:06 TRC     , tsocket.cpp         ,        }exit TSocket::readAbsolut(char *buffer, size_t size) Elapsed time:
27182[ns] --> 0s 0ms
2022-02-05 14:20:06 TRC   625, tamxnet.cpp         ,        {entry TAmxNet::handle_read(const error_code& error, size_t n, R_TOKEN
tk)
2022-02-05 14:20:06 DBG    --,                     , Token: 14, 9 bytes
2022-02-05 14:20:06 DBG    --,                     , Received message type: 0x0007
2022-02-05 14:20:06 TRC   608, tpagemanager.cpp    ,         {entry TPageManager::doCommand(const amx::ANET_COMMAND& cmd)
2022-02-05 14:20:06 TRC   509, tamxcommands.cpp    ,          {entry TAmxCommands::parseCommand(int device, int port, const string&
cmd)
2022-02-05 14:20:06 TRC     ,                     , Parsing for device <10010:14:0> the command: OFF
2022-02-05 14:20:06 TRC   196, tamxcommands.cpp    ,           {entry findCmdDefines(const string& cmd)
2022-02-05 14:20:06 TRC     , tamxcommands.cpp    ,           }exit findCmdDefines(const string& cmd) Elapsed time: 2330[ns] --> 0s
0ms
```

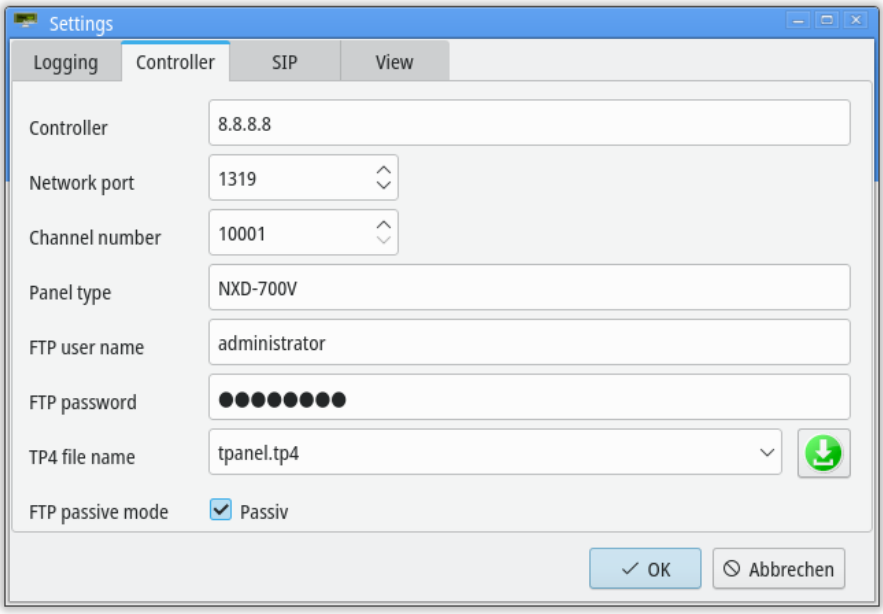
Logfile

This line defines where the log should be written. Select a path and name for the logfile. If you want to see some logs on a mobile device (phone, …) you must make sure that the file is written somewhere in the user space. By default the log file is set to the internal directory where the program itself is stored. This directory is accessible only by the program!

***Note**: On mobile devices you should not activate Trace because this writes a lot of information in a very short time. The program is not only slowed down but it may fill up your memory in the phone very quick.*

Controller

The controller page let you set everything about the controller. There is the IP address the controller is listening on and you may define the FTP credentials to download TP4 files directly from the controller.



Picture 2: Controller settings

Downloading the surface from the controller

From version 1.3.0 on it is possible to put one or more normal TP4 files on the disc of the controller. The name of the files doesn't matter, because the settings dialog is reading the directory on the controller and show all files found with an extension *TP4*. If no TP4 – file was found on the controller, the default file `tpanel.tp4` is set in the settings dialog.

If **TPanel** is started and there is no surface found, it tries to connect to the controller via FTP (File Transfer Protocol) with the credentials defined in the settings. If it succeeds and a file is found, then it downloads the file and unpacks it. Afterward **TPanel** loads the fresh surface and displays it.

If there is already a surface available and the name of the surface file in the settings dialog is changed, it tries to download this new file. If it succeeds it deletes the old surface and unpacks the new one. Afterward it restarts itself.

Controller

Enter in this field either the network name of the controller or the IP address. If this field contains no valid address the program is not able to connect the controller.

***Note**: Currently only plain access is possible. **TPanel** doesn't allow encrypted access to a controller!*

Network port

Enter the network port number the controller is listening on. If not changed in the setup of the controller this is port 1319.

Channel number

Enter the channel number of the panel. This must be a number between 10000 and 19999.

Panel type

Enter here the description of the panel. This should be a name supported by *TPDesign4*. For example a valid name would be MVP-5200 or NXD-700V. Avoid a small “i” at the end because this would mean that the panel can communicate with another one. Currently the panel to panel communication is not implemented in **TPanel**!

FTP user name

This defines the user name of the FTP user. By default this is set to **`administrator`**. The user name is used to automatically login to the controller and look for a TP4 file.

FTP password

This defines the password needed to logon to the controller via FTP. By default this is set to **`password`**. Set this to the password the controller expects.

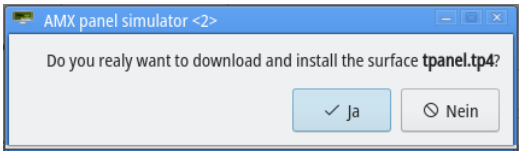
TP4 file name

This defines the name of a TP4 file. Such a file can be downloaded automatically from **TPanel**. If the FTP credentials (user name and password) are correct and there is at least one TP4 file on the disc of the controller, then **TPanel** downloads the file, unpacks it and installs it. After an automatic restart the new surface is visible.

This so called combo box is the combination of an edit line and a list. When the settings dialog is opened, **TPanel** tries to connect to the controller with the given FTP credentials. If it succeeds it reads the root directory of the controller. It puts each file with a file extension of TP4 into this element. When you click on the small arrow on the right end of the line, it opens up and shows the content, if any.

If there were no TP4 files found on the controller then it contains only the default file name `tpanel.tp4`.

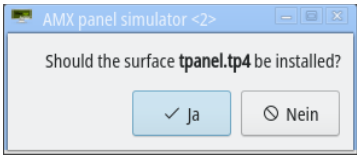
On the right end of this line is a button with a down arrow visible. If you click on it a dialog box opens.



Picture 3: Question to download a file.

If you want to download this file click on YES. Otherwise on NO. If you select YES, then the force button gets a red background. This means that the download starts at the moment the settings dialog is closed.

If you select another surface file and close the dialog, **TPanel** ask you if you want to download the new surface.



Picture 4: Install surface?

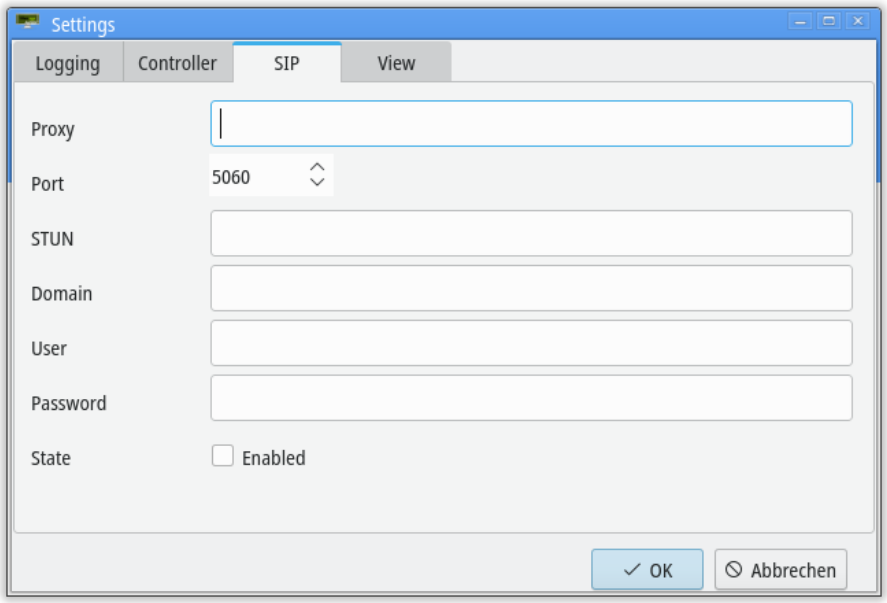
If you click on YES the download starts. Otherwise nothing happens.

FTP passive mode

The FTP protocol knows two different methods to establish a data channel to transfer data from the controller. The default mode is called *port* and requires that your client has full access to the controller. This works as long as there is no firewall in between who blocks the network port 20. To overcome any firewalls, the protocol knows a mode called *passive*. In this case the client connects to a second channel the same way as it did with the control channel. This makes sure that the client can connect even if there is a firewall between. In doubt check this option.

SIP

This is a protocol to connect **TPanel** to a digital phone. It is planned for one of the next releases to implement SIP and the commands reserved for it. *Currently this settings do nothing!*



Picture 5: SIP settings

Proxy

The name or IP address of the SIP server.

Port

The network port the SIP server is listening on. By default this is set to port 5060.

STUN

Sets the IP address for the STUN server

Domain

Sets the realm for authentication.

User

Sets the user name for authentication with the SIP server (proxy address).

Password

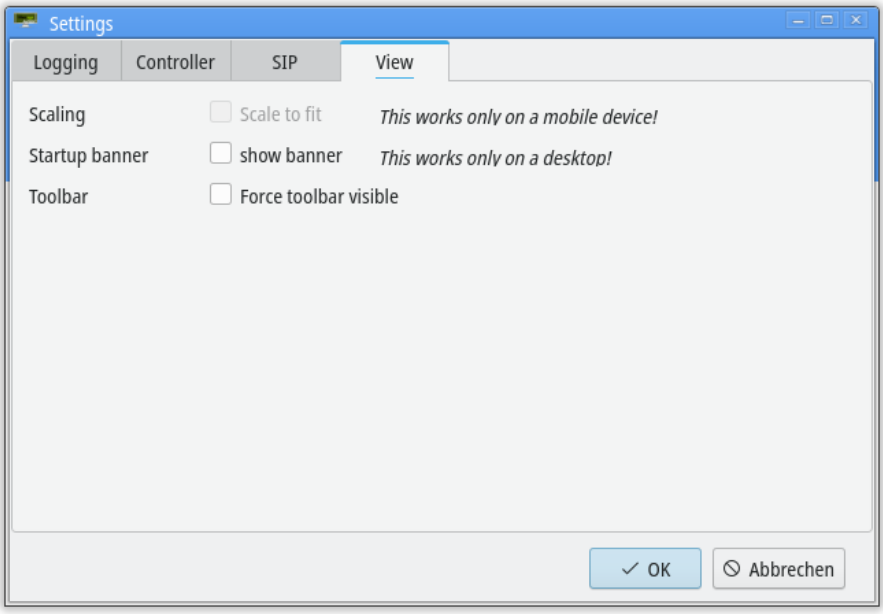
Sets the user password so **TPanel** can connect to the SIP server (SIP proxy server).

State

If this is checked, the SIP settings are enabled and TPanel tries to connect to the SIP proxy server.

View

This tab allows you to set some features of visibility. It allows to select scaling or to force a toolbar to be displayed.



Picture 6: Optical settings

Scaling

On a desktop this is disabled by default and if enabled has no effect.

On a mobile device this is enabled by default and makes sure that the surface fits the size of the display. **TPanel** maintains the aspect ratio which means that you may have a black bar on the left side or at bottom. It depends on the size of the display. If scaling is disabled the real size is used. It depends on the size of the simulated panel (NXD-700Vi has 800 x 480 pixels) and the number of pixels the mobile device offers. For this example we can assume that a mobile device has a higher resolution and therefor the surface would look like very small.

Startup banner

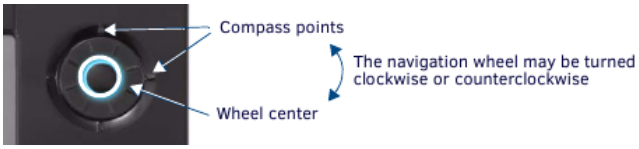
This makes sense only on a desktop. Therefor it is disabled on a mobile device.

If this is checked, TPanel shows a small message on startup from the command line.

```
$ tpanel -c tpanel.cfg
tpanel v1.3.0
(C) Andreas Theofilu <andreas@theosys.at>
This program is under the terms of GPL version 3
```

Toolbar

The toolbar simulates some hard buttons of a real panel. If we take the panel type MVP-5200i for example, we have a round wheel on the right which is also 4 buttons. There is an additional button in the center of the wheel. This buttons are programmable with TPDesign4.



Picture 7: Navigation wheel on a MVP-5200i

Because such buttons are a practical shortcut for navigation, TPanel has a toolbar with some similar functions.

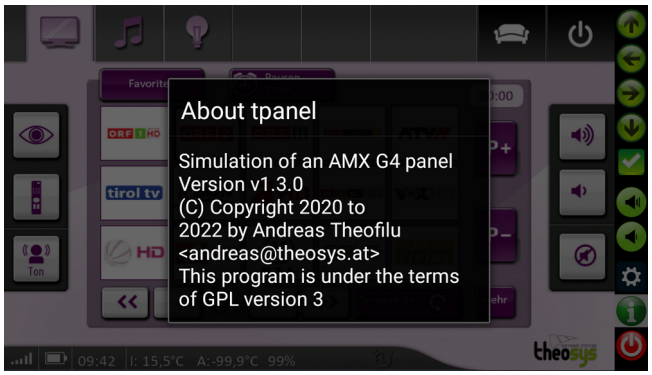
Picture 8: Toolbar of TPanel

As you can see, there are navigation buttons into 4 directions and a select button. Then the toolbar offers 2 buttons to control volume.

Under the volume buttons you find 3 more buttons:

 This opens the settings dialog (look at Setup Dialog at page 10).

 This opens an about dialog:



Picture 9: About dialog on a mobile device

 This ends the program. If you press this button the application ends truly.

Configuration file

To set all aspects necessary **TPanel** allows the use of a configuration file. If **TPanel** is started without any parameters the configuration file is searched in the following locations:

1. `/etc/tpanel.conf`
2. `/etc/tpanel/tpanel.conf`
3. `/usr/etc/tpanel.conf`
4. `/usr/etc/tpanel/tpanel.conf`
5. `$HOME/.tpanel.conf`

On a *Mac* the following additional paths are searched:

1. `/opt/local/etc/tpanel.conf`
2. `/opt/local/etc/tpanel/tpanel.conf`
3. `/opt/local/usr/etc/tpanel.conf`
4. `/opt/local/usr/etc/tpanel/tpanel.conf`

If **TPanel** was not able to find a configuration file in one of the above locations and no command line parameter was given, a default configuration file and storage structure is created in the current directory. This is useful especially on a mobile device where the location of the data directory of the application is accessible only by the application itself.

TPanel offers only one command line parameter "**`-c`**" or "**`--config-file`**". This parameter must be followed by a path and name of a configuration file.

The configuration file itself is a plain text file containing one definition per line. Lines starting with a hash sign (#) or empty lines are ignored. Any valid line consists of the name of the configuration option followed by an equal sign (=) and then the content of the option.

```
LogFile=./tpanel.log
LogLevel=INFO|WARNING|ERROR|TRACE|DEBUG
ProjectPath=/home/andreas/projects/tpanel/tp4.out
NoBanner=true
LongFormat=true
Address=8.8.8.8
Port=1319
Channel=10001
System=0
PanelType=MVP-5200
Firmware=1.0.0
CertCheck=false
Scale=false
Profiling=true
Password1=
Password2=
Password3=
Password4=
SystemSoundFile=singleBeep.wav
SystemSoundState=OFF
SystemSingleBeep=singleBeep01.wav
```

Text 1: Example of a configuration file (shortened)

The following section describes each of the possible configuration options.

LogFile

This defines the path and name of a log file.

TPanel is able to log several information's into a file. It depends on the `LogLevel` what information is logged. ***TPanel*** *must have write permissions to the directory and the defined log file!*

Example:

```
LogFile=/home/user/logs/tpanel.log
```

LogLevel

This defines the log levels. Each level is a level on it's own and may be combined with any other level. The possible levels are:

Level	Description
`INFO`	Logs information's.
`WARNING`	Logs warnings. Some of this warnings could be a an advice for a minor problem.
`ERROR`	Logs errors. There may occur different errors and some of them may be serious.
`TRACE`	Logs tracing messages. This information is mostly useful for programmers who want to know where an error or warning occurred. Each method in every class prints at first a trace message with it's name and another trace message when the method ends. This allows a programmer to follow the flow of the program interanally.
`DEBUG`	Logs a lot of debugging messages mostly useful for programers.
`PROFILE`	This is a special level which combines the levels INFO, WARNING and ERROR. Instead to define the 3 levels it is enough to user this level.
`ALL`	As the name suggests, this enables all levels. Be careful on mobile devices, because this will write a lot of messages in a very short time. It may fill the disc space of a small device very quickly!

Example:

```
LogLevel=INFO|WARNING|ERROR|TRACE
```

LongFormat

Enables or disables the long log format.

By default the long format is disabled. If enabled, each line in the log file starts with a timestamp and contains additional columns containing the line number and the name of the file where the method is located. This is useful only with the log level `TRACE` enabled.

Example:

```
LongFormat=true
```

Profiling

Enables time measuring in the log files.

This is set to *false* by default. If this is set to *true* and *TRACE* log level is enabled all method exit messages show the elapsed time in nanoseconds, milliseconds and seconds.

Example:

```
Profiling=true
```

NoBanner

On a desktop system the program can print a short banner message on startup.

By default this option is enabled. If the program is started from a console (command line) it prints a short banner information:

```
tpanel v1.2.1
(C) Andreas Theofilu <andreas@theosys.at>
This program is under the terms of GPL version 3
```

Text 2: Banner on Startup

If this option is set to *true*, no banner is printed on startup.

Example:

```
NoBanner=false
```

Address

The IP address of the AMX controller.

This option defines the network IP address or the network name of the controller. The address can be a IPv4 or a IPv6 address. The network name, if there is one, is also allowed.

Example:

```
Address=8.8.8.8
```

Port

Defines the network port the controller is listening on.

By default this is set to port 1319. If the controller was configured to any other port number, this parameter must be adapted.

Example:

```
Port=1319
```

Channel

Defines the channel number of the panel.

This parameter is mandatory! The panel number must be in the range of 10000 to 11999. Any other number is invalid. This number must be a unique number on the controller.

Example:

```
Channel=10001
```

System

This defines the system number of the controller the panel connects to.

By default this number is **0**. If there are more than one controller in the network or the controller the panel connects to has any number other than **1**, the corresponding system number should be entered here.

TPanel does currently not support any auto configuration!

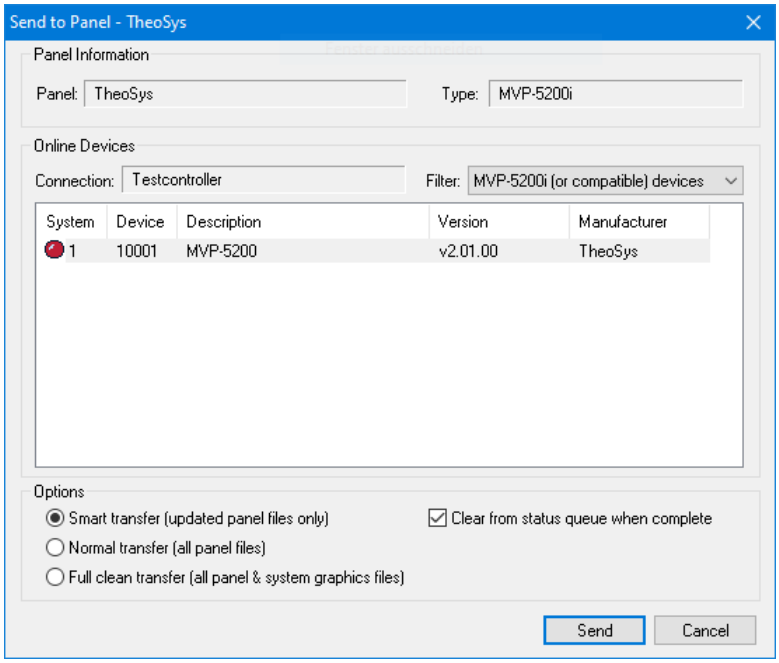
Example:

```
System=0
```

PanelType

Defines the type of the panel **TPanel** claims to be.

By default this is set to *Android*. For *TPDesign4* this is a known name but points to a virtual *TPControl* panel which is not directly accessible from *TPDesign4*. **TPanel** has integrated the same load options as a real AMX panel. This means, that you can transfer the surface over *TPDesign4* as with any real panel. To be able to see the panel in *TPDesign4* define here the name of any real TP4 AMX panel. It is recommended to leave out the small "**i**" at the end of the name because **TPanel** currently does not support panel to panel communication.



Picture 10: TPDesign4 surface transfer (send to panel)

Example:

```
PanelType=MVP-5200
```

ProjectPath

Defines the path where the project files (files for surface of the panel) are on disc.

This option is *mandatory* on desktop systems! On mobile devices it is set to the data location of application.

This directory must not exist at first startup of **TPanel**. But the application must have write rights in the directory. On first startup **TPanel** creates a directory structure in the given path and puts some default files there. Then it displays a system page. With *TPDesign4* it is possible to send the surface to **TPanel** like with any real AMX panel. At first time it is recommended to enable the option **Full clean transfer** because this will also transfer the system panel files. Any later transfer can be done with the default option of *TPDesign4*.

Example:

```
ProjectPath=/usr/share/tpanel
```

Firmware

The internally used firmware version to identify against an AMX controller.

By default this is set to ***1.0.0*** on mobile devices. It is not necessary to change this unless there is a need to. The version number is necessary when the panel connects to a controller and the device **TPanel** is running on is not a desktop.

If **TPanel** is running on a Linux desktop this is set to the version of the Linux kernel.

```
Show Device
-----------

Local devices for system #1 (This System)
------------------------------------------------------------------------------
Device (ID)Model                (ID)Mfg                     FWID  Version
00000  (00299)NI Master         (00001)AMX LLC              00380 v4.1.419
       (PID=0:OID=0) Serial='210504x0700037',0,0           Failed Pings=0
       Physical Address=IP 8.8.8.8:1319 (00:60:9f:94:c4:d5)
         (00299)vxWorks Image   (00001)                     00378 v4.1.419
         (PID=0:OID=1) Serial=N/A
         (00299)BootROM         (00001)                     00379 v4.1.419
         (PID=0:OID=2) Serial=N/A
         (00256)AXLink I/F uContr(00001)                    00270 v1.30.8
         (PID=0:OID=3) Serial=0000000000000000
05001  (00286)NI-2100           (00001)AMX LLC              00383 v1.30.8
       (PID=0:OID=0) Serial='N/A',0,0,0,0,0,0,0,0,0,       Failed Pings=0
       Physical Address=Internal Connection
10001  (00355)NXD-700V          (00001)TheoSys              00656 v2.01.00
       (PID=0:OID=0) Serial=                               Failed Pings=0
       Physical Address=IP 8.8.8.8
         (00355)Kernel          (00001)                     00657 5.15.0-3-amd64
         (PID=0:OID=2) Serial=N/A
33099  (65534)Virtual           (00001)AMX LLC              00380 v4.1.419
       (PID=0:OID=0) Serial='210504x0700037',0,0           Failed Pings=0
       Physical Address=None
```

Text 3: Device connected to a AMX controller

***Note**: This version must not be less then 1.0.0!*

Example:

```
Firmware=1.0.0
```

CertCheck

Evaluates a certificate if downloading from a REST server.

This is set to false by default. If the surface requires to download resources from a WEB server over HTTPS protocol and this is set to true, the certificate of the server is checked. If the certificate is invalid downloading from the source is refused.

Example:

```
CertCheck=true
```

Scale

Enables scaling on mobile devices.

On mobile devices this is set to true by default. On other devices this is ignored. Look at Scaling on page 17 for more information.

Example:

```
Scale=false
```

Password[1-4]

Defines the default password for the protected settings.

This is currently not used!

SystemSoundFile

Defines the system sound file to use when a button is touched.

This is set to `singleBeep.wav` by default. With this setting any sound file in the system section of the settings directory tree can be used. If *`SystemSoundState`* is set to *`true`* this sound is played on every key press or click. To get all the system settings of a AMX panel load the surface the first time with option ***Full clean transfer*** enabled in TPDesign4.

Example:

```
SystemSoundFile=singleBeep.wav
```

SystemSoundState

Defines whether system sounds should be played or not.

This is set to `ON` by default. If the option *`SystemSoundFile`* defines a valid sound file it is played on every push on a button.

Example:

```
SystemSoundState=OFF
```

SystemSingleBeep

Defines the sound file to play if the command `ABEEP` or `BEEP` is received.

Example:

```
SystemSingleBeep=singleBeep01.wav
```

SystemDoubleBeep

Defines the sound file to play if the command A`DBEEP` or `DBEEP` is received.

Example:

```
SystemDoubleBeep=doubleBeep01.wav
```

FTPuser

Defines the FTP user on the controller. This is used to access the controller. By default this is set to `administrator`.

Example:

```
FTPuser=administrator
```

FTPpassword

Defines the FTP password on the controller. This is used to access the controller. By default this is set to `password`.

Example:

```
FTPpassword=password
```

FTPsurface

This defines the file name of a TP4 file on the disc of the controller. If the FTP credentials are correct configured, **TPanel** is able to download the surface directly from the controller. By default this is set to `tpanel.tp4`.

Example:

```
FTPsurface=tpanel.tp4
```

FTPpassive

This defines the way the FTP connection of the data channel is done. If this is set to `false` the normal connection over port 20 is established. This may not work if there is a firewall between the client and the controller. In this case this should be set to `true`. Then the passive mode is used. By default this is set to `true`.

Example:

```
FTPpassive=true
```

FTPdownloadTime

This is an internal used value and must not be set. **TPanel** sets it at the moment it could successfully download a surface file from the controller.

Example:

```
FTPdownloadTime=1646664837
```

SIP_DOMAIN

Currently not implemented

SIP_PROXY

Currently not implemented

SIP_PORT

Currently not implemented

SIP_STUN

Currently not implemented

SIP_USER

Currently not implemented

SIP_PASSWORD

Currently not implemented

SIP_ENABLED

Currently not implemented

Using TPanel on a mobile device

TPanel is currently not available in the *Play Store* of *Android*. Therefor it must be downloaded from my page at https://www.theosys.at. Get the latest version (`tpanel_android_vX.X.X.apk`) and copy it to a location on your mobile device. The device must run with ***Android 10*** or newer to be able to use the application.

Enable APK installs on non Samsung devices

1. Go to your phones **Settings**
2. Go to **Security & privacy** > **More settings**.
3. Tap on **Install apps from external sources**.
4. Select the browser (e.g., Chrome or Firefox) you want to download the APK files from.
5. Toggle **Allow app installs** on.

Enable APK installs on Samsung devices

1. Go to your phone's **Settings**.
2. Go to **Biometrics and security** > **Install unknown apps**.
3. Select the browser (e.g., Chrome or Firefox) you want to download the APK files from.
4. Toggle **Allow app installs** on.

Starting TPanel

Once the application is installed, it can be started. On the first start it presents a green background with my logo on it and 2 buttons. Press either on `Setup`, make a *pinch* gesture or push the *settings button* on the toolbar on right, if present, to get the settings dialog. Look at Setup Dialog on page 10 for detailed information's.

If the setup is completed and the dialog is closed, it takes up to 30 seconds until **TPanel** connects to the controller. Now two scenarios may happen:

- **There is no TP4 file on the controller.**
  After the restart the previous surface (the green one) reappears. Open *TPDesign4* and upload your surface to **TPanel** as you would do for a real panel. The transfer display occurs and shows the progress. At the moment the transfer finished, it takes up to 30 seconds until the new interface appears. Now you can use **TPanel** the same way as a real panel.
- **TP4 file on the controller.**
  Upload a TP4 file to the controller via FTP. Open the setup dialog and enter the name of the file under the tab `Controller` in the field `TP4 file name`. Make sure the FTP user name and the FTP password is correct. Press the button `Ok` and wait. You'll see a busy dialog

during the download and then the application restarts. When it reappears the new surface is visible.

In case of problems

TPanel is far from complete currently. Any command not documented here is not supported or may not work as expected. Therefor it may be that your *NetLinx* program sends commands who are ignored but are mandatory for your surface to work. It is also possible that some commands documented here are not working as expected. *In such cases inform me please!*

Please send an eMail to andreas@theosys.at and attach a short demo program together with a surface file which triggers the error or problem. I will try to fix **TPanel** as fast as possible. In your eMail put the following topics:

- What have I done
- What was expected to happen
- What happened instead

Things who are not working

The following things will not work in the near future because they are proprietary or a secret of AMX I can not reverse engineer.

Passwords in resources

If you've defined a password for a resource, the access to a camera for example, the password is encrypted. Until now I was not able to find out the algorithm used to encrypt it. Therefor this will not work. A workaround could be to open the XML file `prj.xma` and search for the section `resourceList`. There you can find the definitions for the wanted resource. The section may look like:

```
<resource>
   <name>Camera 1</name>
   <protocol>HTTP</protocol>
   <user>user</user>
   <password encrypted="1">2312F21D0E2BA367</password>
   <host>8.8.8.8</host>
   <file>snapshot.cgi</file>
   <refresh>1</refresh>
</resource>
```

Change the line

`<password encrypted="1">2312F21D0E2BA367</password>`

into

`<password encrypted="0">password</password>`

Panel to panel communication

The codec used for communication between panels is proprietary. It is close to a standard but some mandatory parameters are different. I had not the time to investigate in this and my knowledge of this stuff is limited. Therefor this is not supported currently and may not be for a long time.

SIP

This is planned to be supported in the future. But it takes time to implement it. Please be patient.

TakeNotes

I was not able to find out what this is. Because of that it is not supported.

Remote computer control

I plan to implement this for Linux desktops. Because I for myself don’t need it, it has very low priority. So please be patient.

TP5 support

While it is trivial to support the commands (most of them) it is not so easy to read the configuration files. They are encrypted and until today I was not able to find out the algorithm to decrypt them. If I ever find this out, I will support TP5.

In the mean time some of the G5 commands are supported. In short: All commands similar to one of the G4 commands are supported. Included is the command `^BMP` which is the same as the G4 command but has some extensions. This should make it easier for integrators to use **TPanel** mostly the same way as a native G5 panel although the surface is still G4.

Page commands

Page Commands	

@APG	Add a specific popup page to a specified popup group if it does not already exist. If the new popup is added to a group which has a popup displayed on the current page along with the new pop-up, the displayed popup will be hidden and the new popup will be displayed.
	Syntax:
	`"'@APG-<popup page name>;<popup group name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	• popup group name = 1 - 50 ASCII characters. Name of the popup group.
	Example:
	`SEND_COMMAND Panel,"'@APG-Popup1;Group1'"`
	Adds the popup page ’Popup1’ to the popup group ’Group1’.

@CPG	Clear all popup pages from specified popup group.
	
	Syntax:
	`"'@CPG-<popup group name>'"`
	Variable:
	• popup group name = 1 - 50 ASCII characters. Name of the popup group.
	Example:
	`SEND_COMMAND Panel,"'@CPG-Group1'"`
	Clears all popup pages from the popup group ’Group1’.

@DPG	Delete a specific popup page from specified popup group if it exists.
	Syntax:
	`"'@DPG-<popup page name>;<popup group name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	• popup group name = 1 - 50 ASCII characters. Name of the popup group.
	Example:
	`SEND_COMMAND Panel,"'@DPG-Popup1;Group1'"`
	Deletes the popup page ’Popup1’ from the popup group ’Group1’.

@PHE	Set the hide effect for the specified popup page to the named hide effect.
	Syntax:
	`"'@PHE-<popup page name>;<hide effect name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	• hide effect name = Refers to the popup effect names being used.
	Example:
	`SEND_COMMAND Panel,"'@PHE-Popup1;Slide to Left'"`
	Sets the Popup1 hide effect name to ’Slide to Left’.

Page Commands	

@PHP	Set the hide effect position. Only 1 coordinate is ever needed for an effect; however, the command will specify both. This command sets the location at which the effect will end at.
	Syntax:
	`"'@PHP-<popup page name>;<x coordinate>,<y coordinate>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	Example:
	`SEND_COMMAND Panel,"'@PHP-Popup1;75,0'"`
	Sets the Popup1 hide effect x-coordinate value to 75 and the y-coordinate value to 0.

@PHT	Set the hide effect time for the specified popup page.
	Syntax:
	`"'@PHT-<popup page name>;<hide effect time>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	• hide effect time = Given in 1/10ths of a second.
	Example:
	`SEND_COMMAND Panel,"'@PHT-Popup1;50'"`
	Sets the Popup1 hide effect time to 5 seconds.

@PPA
^PPA
	Close all popups on a specified page. If the page name is empty, the current page is used. Same as the ’Clear Page’ command in TPDesign4.
	Syntax:
	`"'@PPA-<page name>'"`
	Variable:
	• page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	Example:
	`SEND_COMMAND Panel,"'@PPA-Page1'"`
	Close all pop-ups on Page1.

@PPF
^PPF
	Deactivate a specific popup page on either a specified page or the current page. If the page name is empty, the current page is used (see example 2). If the popup page is part of a group, the whole group is deactivated. This command works in the same way as the ’Hide Popup’ command in TPDesign4.
	Syntax:
	`"'@PPF-<popup page name>;<page name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	• page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	Example:
	`SEND_COMMAND Panel,"'@PPF-Popup1;Main'"`
	Example 2:
	`SEND_COMMAND Panel,"'@PPF-Popup1'"`
	Deactivates the popup page ’Popup1’ on the current page.

Page Commands	

@PPG
^PPG
	Toggle a specific popup page on either a specified page or the current page. If the page name is empty, the current page is used (see example 2). Toggling refers to the activating/deactivating (On/Off) of a popup page. This command works in the same way as the ’Toggle Popup’ command in TPDesign4.
	Syntax:
	`"'@PPG-<popup page name>;<page name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	• page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	Example:
	`SEND_COMMAND Panel,"'@PPG-Popup1;Main'"`
	Toggles the popup page ’Popup1’ on the ’Main’ page from one state to another (On/Off).
	Example 2:
	`SEND_COMMAND Panel,"'@PPG-Popup1'"`
	Toggles the popup page ’Popup1’ on the current page from one state to another (On/Off).

@PPK
^PPK
	Kill refers to the deactivating (Off) of a popup window from all pages. If the pop-up page is part of a group, the whole group is deactivated. This command works in the same way as the 'Clear Group' command in TPDesign 4.
	Syntax:
	`"'@PPK-<popup page name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	Example:
	`SEND_COMMAND Panel,"'@PPK-Popup1'"`
	Kills the popup page ’Popup1’ on all pages.

@PPM
^PPM
	Set the modality of a specific popup page to Modal or NonModal.
	A Modal popup page, when active, only allows you to use the buttons and features on that popup page. All other buttons on the panel page are inactivated.
	Syntax:
	`"'@PPM-<popup page name>;<mode>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	mode =
	◦ NONMODAL converts a previously Modal popup page to a NonModal.
	◦ MODAL converts a previously NonModal popup page to Modal.
	◦ modal = 1 and non-modal = 0
	Example:
	`SEND_COMMAND Panel,"'@PPM-Popup1;Modal'"`
	Sets the popup page ’Popup1’ to Modal.
	`SEND_COMMAND Panel,"'@PPM-Popup1;1'"`
	Sets the popup page ’Popup1’ to Modal.

Page Commands	

@PPN
^PPN
	Activate a specific popup page to launch on either a specified page or the current page. If the page name is empty, the current page is used (see example 2). If the popup page is already on, do not re-draw it. This command works in the same way as the 'Show Popup' command in TPDesign4.
	Syntax:
	`"'@PPN-<popup page name>;<page name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	• page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	Example:
	`SEND_COMMAND Panel,"'@PPN-Popup1;Main'"`
	Activates 'Popup1' on the 'Main' page.
	Example 2:
	`SEND_COMMAND Panel,"'@PPN-Popup1'"`
	Activates the popup page 'Popup1' on the current page.

@PPT
^PPT
	Set a specific popup page to timeout within a specified time. If timeout is empty, popup page will clear the timeout.
	Syntax:
	`"'@PPT-<popup page name>;<timeout>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	• timeout = Timeout duration in 1/10ths of a second.
	Example:
	`SEND_COMMAND Panel,"'@PPT-Popup1;30'"`
	Sets the popup page 'Popup1' to timeout within 3 seconds.

@PPX
^PPX
	Close all popups on all pages. This command works in the same way as the 'Clear All' command in TPDesign 4.
	Syntax:
	`"'@PPX'"`
	Example:
	`SEND_COMMAND Panel,"'@PPX'"`
	Close all popups on all pages.

@PSE	Set the show effect for the specified popup page to the named show effect.
	Syntax:
	`"'@PSE-<popup page name>;<show effect name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	• show effect name = Refers to the popup effect name being used.
	Example:
	`SEND_COMMAND Panel,"'@PSE-Popup1;Slide from Left'"`
	Sets the Popup1 show effect name to 'Slide from Left'.

Page Commands	
@PSP	Set the show effect position. Only 1 coordinate is ever needed for an effect; however, the command will specify both. This command sets the location at which the effect will begin. **Syntax**: "'@PSP-<popup page name>;<x coordinate>,<y coordinate>'" **Variable**: • popup page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On. **Example**: SEND_COMMAND Panel,"'@PSP-Popup1;100,0'" Sets the Popup1 show effect x-coordinate value to 100 and the y-coordinate value to 0.
@PST	Set the show effect time for the specified popup page. **Syntax**: "'@PST-<popup page name>;<show effect time>'" **Variable**: • popup page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On. • show effect time = Given in 1/10ths of a second. **Example**: SEND_COMMAND Panel,"'@PST-Popup1;50'" Sets the Popup1 show effect time to 5 seconds.
PAGE ^PGE	Flips to a page with a specified page name. If the page is currently active, it will not redraw the page. **Syntax**: "'PAGE-<page name>'" **Variable**: • page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On. **Example**: SEND_COMMAND Panel,"'PAGE-Page1'" Flips to page1.
PPOF	Deactivate a specific popup page on either a specified page or the current page. If the page name is empty, the current page is used (see example 2). If the popup page is part of a group, the whole group is deactivated. This command works in the same way as the 'Hide Popup' command in TPDesign4. **Syntax**: "'PPOF-<popup page name>;<page name>'" **Variable**: • popup page name = 1 - 50 ASCII characters. Name of the popup page. • page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On. **Example**: SEND_COMMAND Panel,"'PPOF-Popup1;Main'" Deactivates the popup page 'Popup1' on the Main page. Example 2: SEND_COMMAND Panel,"'PPOF-Popup1'" Deactivates the popup page 'Popup1' on the current page.

Page Commands	

PPOG	Toggle a specific popup page on either a specified page or the current page. If the page name is empty, the current page is used (see example 2). Toggling refers to the activating/deactivating (On/Off) of a popup page. This command works in the same way as the ’Toggle Popup’ command in TPDesign4.
	Syntax:
	`"'PPOG-<popup page name>;<page name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	• page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	Example:
	`SEND_COMMAND Panel,"'PPOG-Popup1;Main'"`
	Toggles the popup page ’Popup1’ on the Main page from one state to another (On/Off).
	Example 2:
	`SEND_COMMAND Panel,"'PPOG-Popup1'"`
	Toggles the popup page ’Popup1’ on the current page from one state to another (On/Off).

PPON	Activate a specific popup page to launch on either a specified page or the current page. If the page name is empty, the current page is used (see example 2). If the popup page is already On, do not re-draw it. This command works in the same way as the ’Show Popup’ command in TPDesign4.
	Syntax:
	`"'PPON-<popup page name>;<page name>'"`
	Variable:
	• popup page name = 1 - 50 ASCII characters. Name of the popup page.
	• page name = 1 - 50 ASCII characters. Name of the page the popup is displayed On.
	Example:
	`SEND_COMMAND Panel,"'PPON-Popup1; Main'"`
	Activates the popup page ’Popup1’ on the Main page.
	Example 2:
	`SEND_COMMAND Panel,"'PPON-Popup1'"`
	Activates the popup page ’Popup1’ on the current page.

Programming Numbers

The following information provides the programming numbers for colors, fonts, and borders.

Colors can be used to set the colors on buttons, sliders, and pages. The lowest color number represents the lightest color-specific display; the highest number represents the darkest display. For example, 0 represents light red, and 5 is dark red.

RGB Triplets and Names For Basic 88 Colors

RGB Values for all 88 Basic Colors										
Index No.	**Name**	**Red**	**Green**	**Blue**		**Index No.**	**Name**	**Red**	**Green**	**Blue**
0	Very Light Red	255	0	0		45	Medium Aqua	0	80	159
1	Light Red	223	0	0		46	Dark Aqua	0	64	127
2	Red	191	0	0		47	Very Dark Aqua	0	48	95
3	Medium Red	159	0	0		48	Very Light Blue	0	0	255
4	Dark Red	127	0	0		49	Light Blue	0	0	223
5	Very Dark Red	95	0	0		50	Blue	0	0	191
6	Very Light Orange	255	128	0		51	Medium Blue	0	0	159
7	Light Orange	223	112	0		52	Dark Blue	0	0	127
8	Orange	191	96	0		53	Very Dark Blue	0	0	95
9	Medium Orange	159	80	0		54	Very Light Purple	128	0	255
10	Dark Orange	127	64	0		55	Light Purple	112	0	223
11	Very Dark Orange	95	48	0		56	Purple	96	0	191
12	Very Light Yellow	255	255	0		57	Medium Purple	80	0	159
13	Light Yellow	223	223	0		58	Dark Purple	64	0	127
14	Yellow	191	191	0		59	Very Dark Purple	48	0	95
15	Medium Yellow	159	159	0		60	Very Light Magenta	255	0	255
16	Dark Yellow	127	127	0		61	Light Magenta	223	0	223
17	Very Dark Yellow	95	95	0		62	Magenta	191	0	191
18	Very Light Lime	128	255	0		63	Medium Magenta	159	0	159
19	Light Lime	112	223	0		64	Dark Magenta	127	0	127
20	Lime	96	191	0		65	Very Dark Magenta	95	0	95
21	Medium Lime	80	159	0		66	Very Light Pink	255	0	128
22	Dark Lime	64	127	0		67	Light Pink	223	0	112
23	Very Dark Lime	48	95	0		68	Pink	191	0	96
24	Very Light Green	0	255	0		69	Medium Pink	159	0	80
25	Light Green	0	223	0		70	Dark Pink	127	0	64
26	Green	0	191	0		71	Very Dark Pink	95	0	48
27	Medium Green	0	159	0		72	White	255	255	255
28	Dark Green	0	127	0		73	Grey1	238	238	238
29	Very Dark Green	0	95	0		74	Grey3	204	204	204
30	Very Light Mint	0	255	128		75	Grey5	170	170	170
31	Light Mint	0	223	112		76	Grey7	136	136	136
32	Mint	0	191	96		77	Grey9	102	102	102
33	Medium Mint	0	159	80		78	Grey4	187	187	187

RGB Values for all 88 Basic Colors										
Index No.	**Name**	**Red**	**Green**	**Blue**		**Index No.**	**Name**	**Red**	**Green**	**Blue**
34	Dark Mint	0	127	64		79	Grey6	153	153	153
35	Very Dark Mint	0	95	48		80	Grey8	119	119	119
36	Very Light Cyan	0	255	255		81	Grey10	85	85	85
37	Light Cyan	0	223	223		82	Grey12	51	51	51
38	Cyan	0	191	191		83	Grey13	34	34	34
39	Medium Cyan	0	159	159		84	Grey2	221	221	221
40	Dark Cyan	0	127	127		85	Grey11	68	68	68
41	Very Dark Cyan	0	95	95		86	Grey14	17	17	17
42	Very Light Aqua	0	128	255		87	Black	0	0	0
43	Light Aqua	0	112	223		255	TRANSPARENT	99	53	99
44	Aqua	0	96	161						

Font Styles And ID Numbers

Font styles can be used to program the text fonts on buttons, sliders, and pages. The following chart shows the default font type and their respective ID numbers generated by TPDesign4.

Default Font Styles and ID Numbers					
Font ID #	**Font type**	**Size**	**Font ID #**	**Font type**	**Size**
1	Courier New	9	19	Arial	9
2	Courier New	12	20	Arial	10
3	Courier New	18	21	Arial	12
4	Courier New	26	22	Arial	14
5	Courier New	32	23	Arial	16
6	Courier New	18	24	Arial	18
7	Courier New	26	25	Arial	20
8	Courier New	34	26	Arial	24
9	AMX Bold	14	27	Arial	36
10	AMX Bold	20	28	Arial Bold	10
11	AMX Bold	36	29	Arial Bold	8

***NOTE**: Fonts must be imported into a TPDesign4 project file. The font ID numbers are assigned by TPDesign4. These values are also listed in the Generate Programmer's Report.*

Border Styles And Programming Numbers

Border styles can be used to program borders on buttons, sliders, and popup pages.

Border Styles and Programming Numbers			
No.	**Border Styles**	**No.**	**Border Styles**
0 - 1	No border	10 - 11	Picture frame
2	Single line	12	Double line
3	Double line	20	Bevel-S
4	Quad line	21	Bevel-M
5 - 6	Circle 15	22 - 23	Circle 15
7	Single line	24 - 27	Neon inactive-S
8	Double line	40 - 41	Diamond 55
9	Quad line		

Button Query Commands

Button Query commands reply back with a custom event. There will be one custom event for each button/state combination. Each query is assigned a unique custom event type. The following example is for debug purposes only:

```
NetLinx Example: CUSTOM_EVENT[device,Address, Custom event type]
DEFINE_EVENT
      CUSTOM_EVENT[TP,529,1001]     // Text
      CUSTOM_EVENT[TP,529,1002]     // Bitmap
      CUSTOM_EVENT[TP,529,1003]     // Icon
      CUSTOM_EVENT[TP,529,1004]     // Text Justification
      CUSTOM_EVENT[TP,529,1005]     // Bitmap Justification
      CUSTOM_EVENT[TP,529,1006]     // Icon Justification
      CUSTOM_EVENT[TP,529,1007]     // Font
      CUSTOM_EVENT[TP,529,1008]     // Text Effect Name
      CUSTOM_EVENT[TP,529,1009]     // Text Effect Color
      CUSTOM_EVENT[TP,529,1010]     // Word Wrap
      CUSTOM_EVENT[TP,529,1011]     // ON state Border Color
      CUSTOM_EVENT[TP,529,1012]     // ON state Fill Color
      CUSTOM_EVENT[TP,529,1013]     // ON state Text Color
      CUSTOM_EVENT[TP,529,1014]     // Border Name
      CUSTOM_EVENT[TP,529,1015]     // Opacity
{
      Send_String 0,"'ButtonGet Id=',ITOA(CUSTOM.ID),' Type=',ITOA(CUSTOM.TYPE)"
      Send_String 0,"'Flag   =',ITOA(CUSTOM.FLAG)"
      Send_String 0,"'VALUE1 =',ITOA(CUSTOM.VALUE1)"
      Send_String 0,"'VALUE2 =',ITOA(CUSTOM.VALUE2)"
      Send_String 0,"'VALUE3 =',ITOA(CUSTOM.VALUE3)"
      Send_String 0,"'TEXT   =',CUSTOM.TEXT"
      Send_String 0,"'TEXT LENGTH =',ITOA(LENGTH_STRING(CUSTOM.TEXT))"
}
```

All custom events have the following 7 fields:

Custom Event Fields	
Uint Flag	0 means text is a standard string, 1 means Unicode encoded string
slong value1	button state number
slong value2	actual length of string (this is not encoded size)
slong value3	index of first character (usually 1 or same as optional index
string text	the text from the button
text length (string encode)	button text length

These fields are populated differently for each query command. The text length (String Encode) field is not used in any command. These Button Commands are used in NetLinx Studio and are case insensitive:

Button Commands	

^ANI	Run a button animation (in 1/10 second).
	Syntax:
	`"'^ANI-<vt addr range>,<start state>,<end state>,<time>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• start state = Beginning of button state (0= current state).
	• end state = End of button state.
	• time = In 1/10 second intervals.
	Example:
	`SEND_COMMAND Panel,"'^ANI-500,1,25,100'"`
	Runs a button animation at text range 500 from state 1 to state 25 for 10 second.

^APF	Add page flip action to a button if it does not already exist.
	Syntax:
	`"'^APF-<vt addr range>,<page flip action>,<page name>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• page flip action =
	◦ **Stan**[dardPage] - Flip to standard page
	◦ **Prev**[iousPage] - Flip to previous page
	◦ **Show**[Popup] - Show Popup page
	◦ **Hide**[Popup] - Hide Popup page
	◦ **Togg**[lePopup] - Toggle popup state
	◦ **ClearG**[roup] - Clear popup page group from all pages
	◦ **ClearP**[age] - Clear all popup pages from a page with the specified page name
	◦ **ClearA**[ll] - Clear all popup pages from all pages
	• page name = 1 - 50 ASCII characters.
	Example:
	`SEND COMMAND Panel,"'^APF-400,Stan,Main Page'"`
	Assigns a button to a standard page flip with page name 'Main Page'.

^BAT	Append non-unicode text.
	Syntax:
	`"'^BAT-<vt addr range>,<button states range>,<new text>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• new text = 1 - 50 ASCII characters.
	Example:
	`SEND_COMMAND Panel,"'^BAT-520,1,Enter City'"`
	Appends the text 'Enter City' to the button’s OFF state.

Button Commands	

^BAU	Append unicode text. Same format as ^UNI.
	Syntax:
	`"'^BAU-<vt addr range>,<button states range>,<unicode text>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• unicode text = 1 - 50 ASCII characters. Unicode characters must be entered in Hex format.
	Example:
	`SEND_COMMAND Panel,"'^BAU-520,1,00770062'"`
	Appends Unicode text '00770062' to the button's OFF state.

^BCB	Set the border color to the specified color. Only if the specified border color is not the same as the current color.
	Note: Color can be assigned by color name (without spaces), number or R,G,B value (RRGGBB or RRGGBBAA).
	Syntax:
	`"'^BCB-<vt addr range>,<button states range>,<color value>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• color value = Refer to the RGB Triplets and Names For Basic 88 Colors table on page 86 for details.
	Example:
	`SEND_COMMAND Panel,"'^BCB-500.504&510,1,12'"`
	Sets the Off state border color to 12 (Yellow). Colors can be set by Color Numbers, Color name, R,G,B,alpha colors
	(RRGGBBAA) and R, G & B colors values (RRGGBB). Refer to the RGB Triplets and Names For Basic 88 Colors table on page 36.

Button Commands	
?BCB	Get the current border color. **Syntax**: "'?BCB-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type **1011**: ◦ Flag - zero ◦ Value1 - Button state number ◦ Value2 - Actual length of string (should be 9) ◦ Value3 - Zero ◦ Text - Hex encoded color value (ex: #000000FF) ◦ Text length - Color name length (should be 9) **Example**: SEND COMMAND Panel,"'?BCB-529,1'" Gets the button 'OFF state' border color. information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1011 Flag = 0 VALUE1 = 1 VALUE2 = 9 VALUE3 = 0 TEXT = #222222FF TEXT LENGTH = 9
^BCF	Set the fill color to the specified color. Only if the specified fill color is not the same as the current color. *Note: Color can be assigned by color name (without spaces), number or R,G,B value (RRGGBB or RRGGBBAA).* **Syntax**: "'^BCF-<vt addr range>,<button states range>,<color value>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • color value = Refer to the RGB Triplets and Names For Basic 88 Colors table on page 36 for details. **Example**: SEND_COMMAND Panel,"'^BCF-500.504&510.515,1,12'" SEND_COMMAND Panel,"'^BCF-500.504&510.515,1,Yellow'" SEND_COMMAND Panel,"'^BCF-500.504&510.515,1,#F4EC0A63''" SEND_COMMAND Panel,"'^BCF-500.504&510.515,1,#F4EC0A'" Sets the Off state fill color by color number. Colors can be set by Color Numbers, Color name, R,G,B,alpha colors (RRGGBBAA) and R, G & B colors values (RRGGBB).

Button Commands	
?BCF	Get the current fill color. **Syntax**: "'?BCF-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type **1012**: ◦ Flag - Zero ◦ Value1 - Button state number ◦ Value2 - Actual length of string (should be 9) ◦ Value3 - Zero ◦ Text - Hex encoded color value (ex: #000000FF) ◦ Text length - Color name length (should be 9) **Example**: SEND COMMAND Panel,"'?BCF-529,1'" Gets the button 'OFF state' fill color information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1012 Flag = 0 VALUE1 = 1 VALUE2 = 9 VALUE3 = 0 TEXT = #FF8000FF TEXT LENGTH = 9
^BCT	Set the text color to the specified color. Only if the specified text color is not the same as the current color. *Note: Color can be assigned by color name (without spaces), number or R,G,B value (RRGGBB or RRGGBBAA).* **Syntax**: "'^BCT-<vt addr range>,<button states range>,<color value>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • color value = Refer to the RGB Triplets and Names For Basic 88 Colors table on page 86 for details. **Example**: SEND_COMMAND Panel,"'^BCT-500.504&510,1,12'" Sets the Off state border color to 12 (Yellow). Colors can be set by Color Numbers, Color name, R,G,B,alpha colors (RRGGBBAA) and R, G & B colors values (RRGGBB).

Button Commands	
?BCT	Get the current text color. **Syntax**: "'?BCT-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type **1013**: ◦ Flag - Zero ◦ Value1 - Button state number ◦ Value2 - Actual length of string (should be 9) ◦ Value3 - Zero ◦ Text - Hex encoded color value (ex: #000000FF) ◦ Text length - Color name length (should be 9) **Example**: SEND COMMAND Panel,"'?BCT-529,1'" Gets the button 'OFF state' text color information. The result sent to Master would be: ButtonGet Id = 529 Type = 1013 Flag = 0 VALUE1 = 1 VALUE2 = 9 VALUE3 = 0 TEXT = #FFFFFEFF TEXT LENGTH = 9
^BDO	Set the button draw order - Determines what order each layer of the button is drawn. **Syntax**: "'^BDO-<vt addr range>,<button states range>,<1-5><1-5><1-5><1-5><1-5>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • layer assignments = • Fill Layer = 1 • Image Layer = 2 • Icon Layer = 3 • Text Layer = 4 • Border Layer = 5 *Note: The layer assignments are from bottom to top. The default draw order is 12345.* **Example**: SEND_COMMAND Panel,"'^BDO-530,1&2,51432'" Sets the button's variable text 530 ON/OFF state draw order (from bottom to top) to Border, Fill, Text, Icon, and Image. **Example 2**: SEND_COMMAND Panel,"'^BDO-1,0,12345'" Sets all states of a button back to its default drawing order.

Button Commands	

^BFB	Set the feedback type of the button. ONLY works on General-type buttons.
	Syntax:
	`"'^BFB-<vt addr range>,<feedback type>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• feedback type = (None, Channel, Invert, On (Always on), Momentary, and Blink).
	Example:
	`SEND_COMMAND Panel,"'^BFB-500,Momentary'"`
	Sets the Feedback type of the button to 'Momentary'.

Button Commands	

^BMC	Button copy command. Copy attributes of the source button to all the destination buttons. Note that the source is a single button state. Each state must be copied as a separate command. The <codes> section represents what attributes will be copied. All codes are 2 char pairs that can be separated by comma, space, percent or just ran together.
	Syntax:
	`"'^BMC-<vt addr range>,<button states range>,<source port>,<source address>,<source state>,<codes>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• source port = 1 - 100.
	• source address = 1 - 4000.
	• source state = 1 - 256.
	• **codes**:
	BM – Picture/Bitmap
	BR – Border
	CB – Border Color
	CF – Fill Color
	CT – Text Color
	EC – Text effect color
	EF – Text effect
	FT – Font
	IC – Icon
	JB – Bitmap alignment
	JI – Icon alignment
	JT – Text alignment
	OP – Opacity
	SO – Button Sound
	TX – Text
	VI – Video slot ID
	WW – Word wrap on/off
	Example:
	`SEND_COMMAND Panel,"'^BMC-425,1,1,500,1,BR'"`
	or
	`SEND_COMMAND Panel,"'^BMC-425,1,1,500,1,%BR'"`
	Copies the OFF state border of button with a variable text address of 500 onto the OFF state border of button with a variable text address of 425.
	Example 2:
	`SEND_COMMAND Panel,"'^BMC-150,1,1,315,1,%BR%FT%TX%BM%IC%CF%CT'"`
	Copies the OFF state border, font, Text, bitmap, icon, fill color and text color of the button with a variable text address of 315 onto the OFF state border, font, Text, bitmap, icon, fill color and text color of the button with a variable text address of 150.

Button Commands	
^BML	Set the maximum length of the text area button. If this value is set to zero (0), the text area has no max length. The maximum length available is 2000. This is only for a Text area input button and not for a Text area input masking button. **Syntax**: "'^BML-<vt addr range>,<max length>'" **Variable**: • variable text address range = 1 - 4000. • max length = 2000 (0=no max length). **Example**: SEND_COMMAND Panel,"'^BML-500,20'" Sets the maximum length of the text area input button to 20 characters.
^BMP	Assign a picture to those buttons with a defined address range. **Syntax**: "'^BMP-<vt addr range>,<button states range>,<name of bitmap/picture>,[bitmap index], [optional justification]'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • name of bitmap/picture = 1 - 50 ASCII characters. • Optional bitmap index = 0 - 5, the state bitmap index to assign the bitmap. The indexes are defined as: ◦ 0 - Chameleon Image (if present) ◦ 1 - Bitmap 1 (Bitmap) ◦ 2 - Bitmap 2 (Icon) ◦ 3 - Bitmap 3 (Bitmap) ◦ 4 - Bitmap 4 (Bitmap) ◦ 5 - Bitmap 5 (Bitmap) • Optional justification = 0-10 where: ◦ 0 - Absolute position: If absolute justification is set, the next two parameters are the X and Y offset of the bitmap for the referenced index. ◦ 1 - top left ◦ 2 - top center ◦ 3 - top right ◦ 4 - middle left ◦ 5 - middle center ◦ 6 - middle right ◦ 7 - bottom left ◦ 8 - bottom center ◦ 9 - bottom right ◦ 10 - scale to fit ◦ If no justification is specified, the current justification is used. **Example**: SEND_COMMAND Panel,"'^BMP-500.504&510.515,1,bitmap.png'" Sets the OFF state picture for the buttons with variable text ranges of 500-504 & 510-515.

Button Commands	
?BMP	Get the current bitmap name. **Syntax**: "'?BMP-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1custom event type **1002**: ◦ Flag - Zero ◦ Value1 - Button state number ◦ Value2 - Actual length of string ◦ Value3 - Zero ◦ Text - String that represents the bitmap name ◦ Text length - Bitmap name text length (should be 9) **Example**: SEND COMMAND Panel,"'?BMP-529,1'" Gets the button 'OFF state' bitmap information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1002 Flag = 0 VALUE1 = 1 VALUE2 = 9 VALUE3 = 0 TEXT = Buggs.png TEXT LENGTH = 9
^BOP	Set the button opacity. The button opacity can be specified as a decimal between 0 - 255, where zero (0) is invisible and 255 is opaque, or as a HEX code, as used in the color commands by preceding the HEX code with the # sign. In this case, #00 becomes invisible and #FF becomes opaque. If the opacity is set to zero (0), this does not make the button inactive, only invisible. **Syntax**: "'^BOP-<vt addr range>,<button states range>,<button opacity>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • button opacity = 0 (invisible) - 255 (opaque). **Example**: SEND_COMMAND Panel,"'^BOP-500.504&510.515,1,200'" SEND_COMMAND Panel,"'^BOP-500.504&510.515,1,#C8'" Both examples set the opacity of the buttons with the variable text range of 500-504 and 510-515 to 200.

Button Commands	
?BOP	Get the overall button opacity. **Syntax**: "'?BOP-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type 1015: • Flag - Zero • Value1 - Button state number • Value2 - Opacity • Value3 - Zero • Text - Blank • Text length - Zero **Example**: SEND COMMAND Panel,"'?BOP-529,1'" Gets the button 'OFF state' opacity information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1015 Flag = 0 VALUE1 = 1 VALUE2 = 200 VALUE3 = 0 TEXT = TEXT LENGTH = 0
^BOR	Set a border to a specific border style associated with a border value for those buttons with a defined address range. Refer to the Border Styles and Programming Numbers table on page 38 for more information. **Syntax**: "'^BOR-<vt addr range>,<border style name or border value>'" **Variable**: • variable text address range = 1 - 4000. • border style name = Refer to the Border Styles and Programming Numbers table on page 87. • border value = 0 - 41. **Examples**: SEND_COMMAND Panel,"'^BOR-500.504&510.515,10'" Sets the border by number (#10) to those buttons with the variable text range of 500-504 & 510-515. SEND_COMMAND Panel,"'^BOR-500.504&510,AMX Elite -M'" Sets the border by name (AMX Elite) to those buttons with the variable text range of 500-504 & 510-515. The border style is available through the TPDesign4 border-style drop-down list. Refer to the TPD4 Border Styles by Name table on page 38 for more information.

Button Commands	

^BOS	Set the button to display either a Video or Non-Video window.
	Syntax:
	`"'^BOS-<vt addr range>,<button states range>,<video state>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• video state = Video Off = 0 and Video On = 1.
	Example:
	`SEND_COMMAND Panel,"'^BOS-500,1,1'"`
	Sets the button to display video.

^BRD	Set the border of a button state/states. Only if the specified border is not the same as the current border. The border names are available through the TPDesign4 border-name drop-down list.
	Syntax:
	`"'^BRD-<vt addr range>,<button states range>,<border name>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• border name = Refer to the Border Styles and Programming Numbers table on page 87.
	Example:
	`SEND_COMMAND Panel,"'^BRD-500.504&510.515,1&2,Quad Line'"`
	Sets the border by name (Quad Line) to those buttons with the variable text range of 500-504 & 510-515. Refer to the TPD4 Border Styles by Name table on page 38.

Button Commands	
?BRD	Get the current border name. **Syntax**: "'?BRD-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type 1014: • Flag - Zero • Value1 - Button state number • Value2 - Actual length of string • Value3 - Zero • Text - String that represents border name • Text length - Border name length **Example**: SEND COMMAND Panel,"'?BRD-529,1'" Gets the button 'OFF state' border information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1014 Flag = 0 VALUE1 = 1 VALUE2 = 22 VALUE3 = 0 TEXT = Double Bevel Raised -L TEXT LENGTH = 22
^BSM	Submit text for text area buttons. This command causes the text areas to send their text as strings to the NetLinx Master. **Syntax**: "'^BSM-<vt addr range>'" **Variable**: • variable text address range = 1 - 4000. **Example**: SEND_COMMAND Panel,"'^BSM-500'" Submits the text of the text area button.
^BSO	Set the sound played when a button is pressed. If the sound name is blank the sound is then cleared. If the sound name is not matched, the button sound is not changed. **Syntax**: "'^BSO-<vt addr range>,<button states range>,<sound name>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • sound name = (blank - sound cleared, not matched - button sound not changed). **Example**: SEND_COMMAND Panel,"'^BSO-500,1&2,music.wav'" Assigns the sound 'music.wav' to the button Off/On states.

Button Commands	

^BSP	Set the button size and its position on the page.
	Syntax:
	`"'^BSP-<vt addr range>,<left>,<top>,<right>,<bottom>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• left = left side of page.
	• top = top of page.
	• right = right side of page.
	• bottom = bottom of page.
	Example:
	`SEND_COMMAND Panel,"'^BSP-530,left,top'"`
	Sets the button with variable text 530 in the left side top of page.

^BWW	Set the button word wrap feature to those buttons with a defined address range. By default, word-wrap is Off.
	Syntax:
	`"'^BWW-<vt addr range>,<button states range>,<word wrap>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• word wrap = (0=Off and 1=On). Default is Off.
	Example:
	`SEND_COMMAND Panel,"'^BWW-500,1,1'"`
	Sets the word wrap on for the button's Off state.

?BWW	Get the current word wrap flag status.
	Syntax:
	`"'?BWW-<vt addr range>,<button states range>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• custom event type **1010**:
	• Flag - Zero
	• Value1 - Button state number
	• Value2 - 0 = no word wrap, 1 = word wrap
	• Value3 - Zero
	• Text - Blank
	• Text length - Zero
	Example:
	`SEND COMMAND Panel,"'?BWW-529,1'"`
	Gets the button 'OFF state' word wrap flag status information.
	The result sent to the Master would be:
	`ButtonGet Id = 529 Type = 1010`
	`Flag = 0`
	`VALUE1 = 1`
	`VALUE2 = 1`
	`VALUE3 = 0`
	`TEXT =`
	`TEXT LENGTH = 0`

Button Commands	
^CPF	Clear all page flips from a button. **Syntax**: "'^CPF-<vt addr range>'" **Variable**: • variable text address range = 1 - 4000. **Example**: SEND_COMMAND Panel,"'^CPF-500'" Clears all page flips from the button.
^DPF	Delete page flips from button if it already exists. **Syntax**: "'^DFP-<vt addr range>,<actions>,<page name>'" **Variable**: • variable text address range = 1 - 4000. • actions = ◦ **Stan**[dardPage] - Flip to standard page ◦ **Prev**[iousPage] - Flip to previous page ◦ **Show**[Popup] - Show Popup page ◦ **Hide**[Popup] - Hide Popup page ◦ **Togg**[lePopup] - Toggle popup state ◦ **ClearG**[roup] - Clear popup page group from all pages ◦ **ClearP**[age] - Clear all popup pages from a page with the specified page name ◦ **ClearA**[ll] - Clear all popup pages from all pages • page name = 1 - 50 ASCII characters. **Example**: SEND COMMAND Panel,"'^DPF-409,Prev'" Deletes the assignment of a button from flipping to a previous page.
^ENA	Enable or disable buttons with a set variable text range. **Syntax**: "'^ENA-<vt addr range>,<command value>'" **Variable**: • variable text address range = 1 - 4000. • command value = (0= disable, 1= enable) **Example**: SEND_COMMAND Panel,"'^ENA-500.504&510.515,0'" Disables button pushes on buttons with variable text range 500-504 & 510-515.

Button Commands	
^FON	Set a font to a specific Font ID value for those buttons with a defined address range. Font ID numbers are generated by the TPDesign4 programmers report. **Syntax**: "'^FON-<vt addr range>,<button states range>,<font value>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • font value = range = 1 - XXX. Refer to the Default Font Styles and ID Numbers section on page 87. **Example**: SEND_COMMAND Panel,"'^FON-500.504&510.515,1&2,4'" Sets the font size to font ID #4 for the On and Off states of buttons with the variable text range of 500-504 & 510-515. *Note: The Font ID is generated by TPD4 and is located in TPD4 through the Main menu. Panel > Generate Programmer's Report >Text Only Format >Readme.txt.*
?FON	Get the current font index. **Syntax**: "'?FON-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type **1007**: • Flag - Zero • Value1 - Button state number • Value2 - Font index • Value3 - Zero • Text - Blank • Text length - Zero **Example**: SEND COMMAND Panel,"'?FON-529,1'" Gets the button 'OFF state' font type index information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1007 Flag = 0 VALUE1 = 1 VALUE2 = 72 VALUE3 = 0 TEXT = TEXT LENGTH = 0
^GLH	Change the bargraph upper limit. **Syntax**: "'^GLH-<vt addr range>,<bargraph hi>'" **Variable**: • variable text address range = 1 - 4000. • bargraph limit range = 1 - 65535 (bargraph upper limit range). **Example**: SEND_COMMAND Panel,"'^GLH-500,1000'" Changes the bargraph upper limit to 1000.

Button Commands	
^GLL	Change the bargraph lower limit. **Syntax**: "'^GLL-<vt addr range>,<bargraph low>'" **Variable**: • variable text address range = 1 - 4000. • bargraph limit range = 1 - 65535 (bargraph lower limit range). **Example**: SEND_COMMAND Panel,"'^GLL-500,150'" Changes the bargraph lower limit to 150.
^GSC	Change the bargraph slider color or joystick cursor color. A user can also assign the color by Name and R,G,B value (RRGGBB or RRGGBBAA). **Syntax**: "'^GSC-<vt addr range>,<color value>'" **Variable**: • variable text address range = 1 - 4000. • color value = Refer to the RGB Triplets and Names For Basic 88 Colors table on page 36. **Example**: SEND_COMMAND Panel,"'^GSC-500,12'" Changes the bargraph or joystick slider color to Yellow.
^ICO	Set the icon to a button. **Syntax**: "'^ICO-<vt addr range>,<button states range>,<icon index>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • icon index range = 0 - 9900 (a value of 0 is clear). **Example**: SEND_COMMAND Panel,"'^ICO-500.504&510.515,1&2,1'" Sets the icon for On and Off states for buttons with variable text ranges of 500-504 & 510-515.

<table>
<tr><th colspan="2">Button Commands</th></tr>
<tr><td>?ICO</td><td>Get the current icon index.
Syntax:
"'?ICO-<vt addr range>,<button states range>'"
Variable:
• variable text address range = 1 - 4000.
• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
• custom event type 1003:
• Flag - Zero
• Value1 - Button state number
• Value2 - Icon Index
• Value3 - Zero
• Text - Blank
• Text length - Zero
Example:
SEND COMMAND Panel,"'?ICO-529,1&2'"
Gets the button 'OFF state' icon index information.
The result sent to the Master would be:
ButtonGet Id = 529 Type = 1003
Flag = 0
VALUE1 = 2
VALUE2 = 12
VALUE3 = 0
TEXT =
TEXT LENGTH = 0</td></tr>
<tr><td>^JSB</td><td>Set bitmap/picture alignment using a numeric keypad layout for those buttons with a defined address range. The alignment of 0 is followed by ',<left>,<top>'. The left and top coordinates are relative to the upper left corner of the button.
Syntax:
"'^JSB-<vt addr range>,<button states range>,<new text alignment>'"
Variable:
• variable text address range = 1 - 4000.
• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
• new text alignment = Value of 1- 9 corresponds to the following locations:
0 (zero can be used for an absolute position)
<table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>
Example:
SEND_COMMAND Panel,"'^JSB-500.504&510.515,1&2,1'"
Sets the off/on state picture alignment to upper left corner for those buttons with variable text ranges of 500-504 & 510-515.</td></tr>
</table>

<table>
<tr><th colspan="2">Button Commands</th></tr>
<tr><td>?JSB</td><td>Get the current bitmap justification.
Syntax:
"'?JSB-<vt addr range>,<button states range>'"
Variable:
• variable text address range = 1 - 4000.
• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
• custom event type 1005:
• Flag - Zero
• Value1 - Button state number
• Value2 - 1 - 9 justify
• Value3 - Zero
• Text - Blank
• Text length - Zero
Example:
SEND_COMMAND Panel,"'?JSB-529,1'"
Gets the button 'OFF state' bitmap justification information.
The result sent to the Master would be:
ButtonGet Id = 529 Type = 1005
Flag = 0
VALUE1 = 1
VALUE2 = 5
VALUE3 = 0
TEXT =
TEXT LENGTH = 0</td></tr>
<tr><td>^JSI</td><td>Set icon alignment using a numeric keypad layout for those buttons with a defined address range. The alignment of 0 is followed by ',<left>,<top>'. The left and top coordinates are relative to the upper left corner of the button.
Syntax:
"'^JSI-<vt addr range>,<button states range>,<new icon alignment>'"
Variable:
• variable text address range = 1 - 4000.
• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
• new icon alignment = Value of 1 - 9 corresponds to the following locations:
0 (zero can be used for an absolute position)
<table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>
Example:
SEND_COMMAND Panel,"'^JSI-500.504&510.515,1&2,1'"
Sets the Off/On state icon alignment to upper left corner for those buttons with variable text range of 500-504 & 510-515.</td></tr>
</table>

<table>
<tr><th colspan="2">Button Commands</th></tr>
<tr><td>?JSI</td><td>Get the current icon justification.
Syntax:
"'?JSI-<vt addr range>,<button states range>'"
Variable:
• variable text address range = 1 - 4000.
• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
• custom event type 1006:
• Flag - Zero
• Value1 - Button state number
• Value2 - 1 - 9 justify
• Value3 - Zero
• Text - Blank
• Text length - Zero
Example:
SEND COMMAND Panel,"'?JSI-529,1'"
Gets the button 'OFF state' icon justification information.
The result sent to the Master would be:
ButtonGet Id = 529 Type = 1006
Flag = 0
VALUE1 = 1
VALUE2 = 6
VALUE3 = 0
TEXT =
TEXT LENGTH = 0</td></tr>
<tr><td>^JST</td><td>Set text alignment using a numeric keypad layout for those buttons with a defined address range. The alignment of 0 is followed by ',<left>,<top>'. The left and top coordinates are relative to the upper left corner of the button.
Syntax:
"'^JST-<vt addr range>,<button states range>,<new text alignment>'"
Variable:
• variable text address range = 1 - 4000.
• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
• new text alignment = Value of 1 - 9 corresponds to the following locations:
0 (zero can be used for an absolute position)
<table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>
Example:
SEND_COMMAND Panel,"'^JST-500.504&510.515,1&2,1'"
Sets the text alignment to the upper left corner for those buttons with variable text ranges of 500-504 & 510-515.</td></tr>
</table>

Button Commands	
?JST	Get the current text justification. **Syntax**: "'?JST-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type **1004**: • Flag - Zero • Value1 - Button state number • Value2 - 1 - 9 justify • Value3 - Zero • Text - Blank • Text length - Zero **Example**: SEND COMMAND Panel,"'?JST-529,1'" Gets the button 'OFF state' text justification information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1004 Flag = 0 VALUE1 = 1 VALUE2 = 1 VALUE3 = 0 TEXT = TEXT LENGTH = 0
^SHO	Show or hide a button with a set variable text range. **Syntax**: "'^SHO-<vt addr range>,<command value>'" **Variable**: • variable text address range = 1 - 4000. • command value = (0= hide, 1= show). **Example**: SEND_COMMAND Panel,"'^SHO-500.504&510.515,0'" Hides buttons with variable text address range 500-504 & 510-515.
^TEC	Set the text effect color for the specified addresses/states to the specified color. The Text Effect is specified by name and can be found in TPD4. You can also assign the color by name or RGB value (RRGGBB or RRGGBBAA). **Syntax**: "'^TEC-<vt addr range>,<button states range>,<color value>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • color value = Refer to the RGB Triplets and Names For Basic 88 Colors table on page 36. **Example**: SEND_COMMAND Panel,"'^TEC-500.504&510.515,1&2,12'" Sets the text effect color to Very Light Yellow on buttons with variable text 500-504 and 510-515.

Button Commands	
?TEC	Get the current text effect color. **Syntax**: "'?TEC-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type **1009**: • Flag - Zero • Value1 - Button state number • Value2 - Actual length of string (should be 9) • Value3 - Zero • Text - Hex encoded color value (ex: #000000FF) • Text length - Color name length **Example**: SEND COMMAND Panel,"'?TEC-529,1'" Gets the button 'OFF state' text effect color information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1009 Flag = 0 VALUE1 = 1 VALUE2 = 9 VALUE3 = 0 TEXT = #5088F2AE TEXT LENGTH = 9
^TEF	Set the text effect. The Text Effect is specified by name and can be found in TPD4. **Syntax**: "'^TEF-<vt addr range>,<button states range>,<text effect name>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • text effect name = Refer to the Text Effects table on page Fehler: Verweis nicht gefunden for a listing of text effect names. **Example**: SEND_COMMAND Panel,"'^TEF-500.504&510.515,1&2,Soft Drop Shadow 3'" Sets the text effect to Soft Drop Shadow 3 for the button with variable text range 500-504 and 510-515.

Button Commands	
?TEF	Get the current text effect name. **Syntax**: "'?TEF-<vt addr range>,<button states range>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • custom event type **1008**: ◦ Flag - Zero ◦ Value1 - Button state number ◦ Value2 - Actual length of string ◦ Value3 - Zero ◦ Text - String that represents the text effect name ◦ Text length - Text effect name length **Example**: SEND_COMMAND Panel,"'?TEF-529,1'" Gets the button 'OFF state' text effect name information. The result sent to the Master would be: ButtonGet Id = 529 Type = 1008 Flag = 0 VALUE1 = 1 VALUE2 = 18 VALUE3 = 0 TEXT = Hard Drop Shadow 3 TEXT LENGTH = 18
^TXT	Assign a text string to those buttons with a defined address range.Sets Non-Unicode text. **Syntax**: "'^TXT-<vt addr range>,<button states range>,<new text>'" **Variable**: • variable text address range = 1 - 4000. • button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state). • new text = 1 - 50 ASCII characters. **Example**: SEND_COMMAND Panel,"'^TXT-500.504&510.515,1&2,Test Only'" Sets the On and Off state text for buttons with the variable text ranges of 500-504 & 510-515.

Button Commands	

?TXT	Get the current text information.
	Syntax:
	`"'?TXT-<vt addr range>,<button states range>,<optional index>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• optional index = This is used if a string was too long to get back in one command. The reply will start at this index.
	• custom event type **1001**:
	◦ Flag - Zero
	◦ Value1 - Button state number
	◦ Value2 - Actual length of string
	◦ Value3 - Index
	◦ Text - Text from the button
	◦ Text length - Button text length
	Example:
	`SEND_COMMAND Panel,"'?TXT-529,1'"`
	Gets the button 'OFF state' text information.
	The result sent to the Master would be:
	`ButtonGet Id = 529 Type = 1001`
	`Flag = 0`
	`VALUE1 = 1`
	`VALUE2 = 14`
	`VALUE3 = 1`
	`TEXT = This is a test`
	`TEXT LENGTH = 14`

^UNI	Set Unicode text in the legacy G4 format. For the ^UNI command, the Unicode text is sent as ASCII-HEX nibbles.
	***Note**: In the legacy format, Unicode text is always represented in a HEX value.*
	Refer to the TPDesign Instruction Manual for more information.
	Syntax:
	`"'^UNI-<addr range>,<button states range>,<unicode text>'"`
	Variables:
	• address range: Address codes of buttons to affect. A '.' between addresses includes the range, and & between addresses includes each address.
	• button states range: 1 - 256 for multi-state buttons (0 = All states, for General buttons, 1 = Off state and 2 = On state).
	• unicode text: Unicode HEX value.
	Example:
	`SEND_COMMAND Panel,"'^UNI-500,1,0041'"`
	Sets the button's unicode character to 'A'.
	`SEND_COMMAND TP,"'^UNI-1,0,0041'"`
	Send the variable text 'A' in unicode to all states of the variable text button 1, (for which the character code is 0041 Hex).

Button Commands	

^UTF	Set button state text using UTF-8 text command - Set State Text Command using UTF-8. Assign a text string encoded with UTF-8 (which is ASCII-compatible) to those buttons with a defined address range.
	While UTF-8 is ASCII compatible, extended ASCII characters in the range 128-255 will be encoded differently based on UTF-8. This command also supports Unicode characters using UTF-8 (which is the encoding method used in >80% of web servers), making the old AMX Hex quad Unicode encoding obsolete.
	Syntax:
	`"'^UTF-<vt addr range>,<button states range>,<new text>'"`
	Variables:
	• variable text address range = 1 - 4000.
	• Button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• unicode text: Unicode UTF-8 text.
	Example:
	`SEND_COMMAND Panel,"'^UTF-500.504&510.515,1&2, ASCII ExtendedASCIIÇüéâäàåç Unicode 動き始めました ' "`
	Sets the On and Off state text for buttons with the variable text ranges of 500-504 & 510-515.

Text Effect Names

The following is a listing of text effects names associated with the ^TEF command.

Text Effects		
Glow -S	Medium Drop Shadow 1	Hard Drop Shadow 1
Glow -M	Medium Drop Shadow 2	Hard Drop Shadow 2
Glow -L	Medium Drop Shadow 3	Hard Drop Shadow 3
Glow -X	Medium Drop Shadow 4	Hard Drop Shadow 4
Outline -S	Medium Drop Shadow 5	Hard Drop Shadow 5
Outline -M	Medium Drop Shadow 6	Hard Drop Shadow 6
Outline -L	Medium Drop Shadow 7	Hard Drop Shadow 7
Outline -X	Medium Drop Shadow 8	Hard Drop Shadow 8
Soft Drop Shadow 1	Medium Drop Shadow 1 with outline	Hard Drop Shadow 1 with outline
Soft Drop Shadow 2	Medium Drop Shadow 2 with outline	Hard Drop Shadow 2 with outline
Soft Drop Shadow 3	Medium Drop Shadow 3 with outline	Hard Drop Shadow 3 with outline
Soft Drop Shadow 4	Medium Drop Shadow 4 with outline	Hard Drop Shadow 4 with outline
Soft Drop Shadow 5	Medium Drop Shadow 5 with outline	Hard Drop Shadow 5 with outline
Soft Drop Shadow 6	Medium Drop Shadow 6 with outline	Hard Drop Shadow 6 with outline
Soft Drop Shadow 7	Medium Drop Shadow 7 with outline	Hard Drop Shadow 7 with outline
Soft Drop Shadow 8	Medium Drop Shadow 8 with outline	Hard Drop Shadow 8 with outline
Soft Drop Shadow 1 with outline		
Soft Drop Shadow 2 with outline		
Soft Drop Shadow 3 with outline		
Soft Drop Shadow 4 with outline		
Soft Drop Shadow 5 with outline		
Soft Drop Shadow 6 with outline		
Soft Drop Shadow 7 with outline		
Soft Drop Shadow 8 with outline		

Panel Runtime Operations

Serial Commands are used in Terminal Emulator mode. These commands are case insensitive.

Panel Runtime Operation Commands	

@AKB	Pop up the keyboard icon and initialize the text string to that specified. Keyboard string is set to null on start up and is stored until the program ends. The Prompt Text is optional.
	Syntax:
	`"'@AKB-<initial text>;<prompt text>'"`
	Variables:
	• initial text = 1 - 50 ASCII characters.
	• prompt text = 1 - 50 ASCII characters.
	Example:
	`SEND_COMMAND Panel,"'@AKB-Texas;Enter State'"`
	Pops up the Keyboard and initializes the text string 'Texas' with prompt text 'Enter State'.

AKEYB	Pop up the keyboard icon and initialize the text string to that specified. Keyboard string is set to null on start up and is stored until the program ends.
	Syntax:
	`"'AKEYB-<initial text>'"`
	Variables:
	`initial text = 1 - 50 ASCII characters.`
	Example:
	`SEND_COMMAND Panel,"'AKEYB-This is a Test'"`
	Pops up the Keyboard and initializes the text string 'This is a Test'.

AKEYP	Pop up the keypad icon and initialize the text string to that specified. The keypad string is set to null on start up and is stored until the program ends.
	Syntax:
	`"'AKEYP-<number string>'"`
	Variables:
	• number string = 0 - 9999.
	Example:
	`SEND_COMMAND Panel,"'AKEP-12345'"`
	Pops up the Keypad and initializes the text string '12345'.

AKEYR	Remove the Keyboard/Keypad.Remove keyboard or keypad that was displayed using 'AKEYB', 'AKEYP', 'PKEYP', @AKB, @AKP, @PKP, @EKP, or @TKP commands.
	Syntax:
	`"'AKEYR'"`
	Example:
	`SEND_COMMAND Panel,"'AKEYR'"`
	Removes the Keyboard/Keypad.

Panel Runtime Operation Commands	
@AKP	Pop up the keypad icon and initialize the text string to that specified. Keypad string is set to null on start up and is stored until the program ends. The Prompt Text is optional. **Syntax**: "'@AKP-<initial text>;<prompt text>'" **Variables**: • initial text = 1 - 50 ASCII characters. • prompt text = 1 - 50 ASCII characters. **Example**: SEND_COMMAND Panel,"'@AKP-12345678;ENTER PASSWORD'" Pops up the Keypad and initializes the text string '12345678' with prompt text 'ENTER PASSWORD'.
@AKR	Remove keyboard or keypad that was displayed using 'AKEYB', 'AKEYP', 'PKEYP', @AKB, @AKP, @PKP, @EKP, or @TKP commands. **Syntax**: "'@AKR'" **Example**: SEND_COMMAND Panel,"'@AKR'" Removes the Keyboard/Keypad.
ABEEP	Output a single beep even if beep is Off. **Syntax**: "'ABEEP'" **Example**: SEND_COMMAND Panel,"'ABEEP'" Outputs a beep of duration 1 beep even if beep is Off.
ADBEEP	Output a double beep even if beep is Off. **Syntax**: "'ADBEEP'" **Example**: SEND_COMMAND Panel,"'ADBEEP'" Outputs a double beep even if beep is Off.
BEEP ^ABP	Output a beep. **Syntax**: "'BEEP'" **Example**: SEND_COMMAND Panel,"'BEEP'" Outputs a beep.
DBEEP ^ADB	Output a double beep. **Syntax**: "'DBEEP'" **Example**: SEND_COMMAND Panel,"'DBEEP'" Outputs a double beep.

Panel Runtime Operation Commands	
@EKP	Extend the Keypad - Pops up the keypad icon and initializes the text string to that specified. The Prompt Text is optional. **Syntax**: "'@EKP-<initial text>;<prompt text>'" **Variables**: • initial text = 1 - 50 ASCII characters. • prompt text = 1 - 50 ASCII characters. **Example**: SEND_COMMAND Panel,"'@EKP-33333333;Enter Password'" Pops up the Keypad and initializes the text string '33333333' with prompt text 'Enter Password'.
PKEYP	Present a private keypad - Pops up the keypad icon and initializes the text string to that specified. Keypad displays a '*' instead of the numbers typed. The Prompt Text is optional. **Syntax**: "'PKEYP-<initial text>'" **Variables**: • initial text = 1 - 50 ASCII characters. **Example**: SEND COMMAND Panel,"'PKEYP-123456789'" Pops up the Keypad and initializes the text string '123456789' in '*'.
@PKP	Present a private keypad - Pops up the keypad icon and initializes the text string to that specified. Keypad displays a '*' instead of the numbers typed. The Prompt Text is optional. **Syntax**: "'@PKP-<initial text>;<prompt text>'" **Variables**: • initial text = 1 - 50 ASCII characters. • prompt text = 1 - 50 ASCII characters. **Example**: SEND COMMAND Panel,"'@PKP-1234567;ENTER PASSWORD'" Pops up the Keypad and initializes the text string 'ENTER PASSWORD' in '*'.
SETUP ^STP	Send panel to SETUP page. **Syntax**: "'SETUP'" **Example**: SEND COMMAND Panel,"'SETUP'" Sends the panel to the Setup Page.
SHUTDOWN	Shut down the program. **Syntax**: "'SHUTDOWN'" **Example**: SEND COMMAND Panel,"'SHUTDOWN'" Ends the application.

Panel Runtime Operation Commands	
@SOU ^SOU	Play a sound file. **Syntax**: "'@SOU-<sound name>'" **Variables**: • sound name = Name of the sound file. Supported sound file formats are: WAV & MP3. **Example**: SEND COMMAND Panel,"'@SOU-Music.wav'" Plays the 'Music.wav' file.
@TKP ^TKP	Present a telephone keypad - Pops up the keypad icon and initializes the text string to that specified. The Prompt Text is optional. **Syntax**: "'@TKP-<initial text>;<prompt text>'" **Variables**: • initial text = 1 - 50 ASCII characters. • prompt text = 1 - 50 ASCII characters. **Example**: SEND COMMAND Panel,"'@TKP-999.222.1211;Enter Phone Number'" Pops-up the Keypad and initializes the text string '999.222.1211' with prompt text 'Enter Phone Number'.
@VKB	Popup the virtual keyboard. **Syntax**: "'@VKB'" **Example**: SEND COMMAND Panel,"'@VKB'" Pops-up the virtual keyboard.

Input Commands

These Send Commands are case insensitive.

Input Commands	
^KPS	Set the keyboard passthru. **Syntax**: "'^KPS-<pass data>'" **Variable**: **pass data**: • <blank/empty> = Disables the keyboard. • 0 = Pass data to G4 application (default). This can be used with VPC or text areas. • 1 - 4 = Not used. • 5 = Sends out data to the Master. **Example**: SEND_COMMAND Panel,"'^KPS-5'" Sets the keyboard passthru to the Master. Option 5 sends keystrokes directly to the Master via the Send Output String mechanism. This process sends a virtual keystroke command (^VKS) to the Master. **Example 2**: SEND_COMMAND Panel,"'^KPS-0'" Disables the keyboard passthru to the Master.
^VKS	Send one or more virtual key strokes to the G4 application. Key presses and key releases are not distinguished except in the case of CTRL, ALT, and SHIFT. Refer to the Embedded Codes table on page 115 that defines special characters which can be included with the string but may not be represented by the ASCII character set. **Syntax**: "'^VKS-<string>'" **Variable**: • string = Only 1 string per command/only one stroke per command. **Example**: SEND_COMMAND Panel,"'^VKS-'8" Sends out the keystroke 'backspace' to the G4 application.

Daynamic Image Commands

The following table describes Dynamic Image Commands.

Dynamic Image Commands	

^BBR	Set the bitmap of a button to use a particular resource.
	Syntax:
	`"'^BBR-<vt addr range>,<button states range>,<resource name>'"`
	Variable:
	• variable text address range = 1 - 4000.
	• button states range = 1 - 256 for multi-state buttons (0 = All states, for General buttons 1 = Off state and 2 = On state).
	• resource name = 1 - 50 ASCII characters.
	Example:
	`SEND_COMMAND Panel,"'^BBR-700,1,Sports_Image'"`
	Sets the resource name of the button to ’Sports_Image’.

^RAF	Add new resources - Adds any and all resource parameters by sending embedded codes and data. Since the embedded codes are preceded by a '%' character, any '%' character contained in the URL must be escaped with a second '%' character (see example).
	The file name field (indicated by a %F embedded code) may contain special escape sequences as shown in the ^RAF, ^RMF - *Embedded Codes* table below.
	Syntax:
	`"'^RAF-<resource name>,<data>'"`
	Variables:
	• resource name = 1 - 50 ASCII characters.
	• data = Refers to the embedded codes, see the ^RAF, ^RMF
	Example:
	`SEND_COMMAND Panel,"'^RAF-New Image,%P0%HAMX.COM%Alab/Test/file%Ftest.jpg'"`
	Adds a new resource.
	The resource name is ’New Image’
	%P (protocol) is an HTTP
	%H (host name) is AMX.COM
	%A (file path) is Lab/Test_f ile
	%F (file name) is test.jpg.

^RFR	Force a refresh for a given resource.
	Syntax:
	`"'^RFR-<resource name>'"`
	Variable:
	• resource name = 1 - 50 ASCII characters.
	Example:
	`SEND_COMMAND Panel,"'^RFR-Sports_Image'"`
	Forces a refresh on ’Sports_Image’.

Dynamic Image Commands	

^RMF	Modify an existing resource - Modifies any and all resource parameters by sending embedded codes and data. Since the embedded codes are preceded by a '%' character, any '%' character contained in the URL must be escaped with a second '%' character (see example).
	The file name field (indicated by a %F embedded code) may contain special escape sequences as shown in the ^RAF, ^RMF
	Syntax:
	`"'^RMF-<resource name>,<data>'"`
	Variables:
	• resource name = 1 - 50 ASCII characters
	• data = Refers to the embedded codes, see the ^RAF, ^RMF
	Example:
	`SEND_COMMAND Panel,"'^RMF-Sports_Image,%ALab/Test/Images%Ftest.jpg'"`
	Changes the resource ’Sports_Image’ file name to ’test.jpg’ and the path to ’Lab_Test/Images’.

^RSR	Change the refresh rate for a given resource.
	Syntax:
	`"'^RSR-<resource name>,<refresh rate>'"`
	Variable:
	• resource name = 1 - 50 ASCII characters.
	• refresh rate = Measured in seconds.
	Example:
	`SEND_COMMAND Panel,"'^RSR-Sports_Image,5'"`
	Sets the refresh rate to 5 seconds for the given resource (’Sports_Image’).