MIPS

Host Application

User Manual

Version 2.22 — March 2026

GAA Custom Electronics, LLC
www.GAACustom.com
gaa@gaa-ce.com

This document was prepared with the assistance of Claude, an AI assistant developed by Anthropic. Technical content is derived from the MIPS application source code and official documentation.

Table of Contents

1. Introduction

The MIPS (Module Intelligent Power Supplies) host application is a Qt-based desktop application developed by GAA Custom Electronics, LLC. It communicates with one or more MIPS (Modular Instrument Platform System) hardware controllers, providing full monitoring and control from a host computer running macOS or Windows.

MIPS controllers are used in scientific instruments — particularly mass spectrometry and ion mobility systems — to generate and control high-voltage DC bias outputs, RF drive signals, traveling-wave ion guides, FAIMS separation fields, pulse sequences, and other instrument functions.

The application supports two primary interaction modes:

- Tab-based module control: When connected, the application presents tabs corresponding to detected hardware modules, allowing direct parameter monitoring and control.
- Custom Control Panels: User-defined graphical panels that organize MIPS controls in a layout matching a specific instrument or workflow, with full scripting support.

1.1 Key Capabilities

- Connect via USB (virtual COM port), Wi-Fi, or Ethernet (TCP/IP)
- Manage multiple MIPS systems simultaneously
- Monitor and control DC bias voltages (up to 32 channels)
- Control RF driver frequencies, drive levels, and auto-tuning
- Control Traveling Wave (Twave) and ARB waveform generators
- Operate FAIMS systems including CV parking and scanning
- Generate complex pulse sequences
- High-speed ADC data acquisition (up to 1 MHz)
- Build fully custom graphical control panels from text configuration files
- Automate instrument operation with embedded ECMAScript scripting
- Remote control via TCP server interface
- Upload new firmware to the MIPS controller

2. Installation

2.1 macOS

1. Copy MIPS.app to your Applications folder or any desired location.
2. Double-click MIPS.app to launch.
3. Copy the MIPS examples folder (inside the app bundle) to your home directory for access to example control panels and scripts.

> **Note:** No USB drivers are required on macOS. The system uses Apple's built-in CDC serial driver.

2.2 Windows

4. Copy the MIPS PC directory (including all subdirectories) to your PC and name it MIPS.
5. Run MIPS.exe. If you see a missing MSVCR110.dll error, run vcredist_x86.exe (included) to install the required Visual C++ runtime.
6. Install the Arduino Due USB driver. Follow the guide at https://www.arduino.cc/en/Guide/ArduinoDue and point the installer to the drivers folder inside the MIPS directory.
7. Launch MIPS.exe and proceed to connect to your hardware.

> **Note:** Install files for both platforms are available on the GAA Custom Electronics Google Drive: https://drive.google.com/open?id=0B3IchPRNYqYIcjZhdGFVMVR1VzQ

3. Getting Started — Connecting to MIPS

When MIPS launches, the System tab is shown with no hardware connected. The right panel displays the connection instructions. Use the buttons on the left to establish a connection before using any other tab.

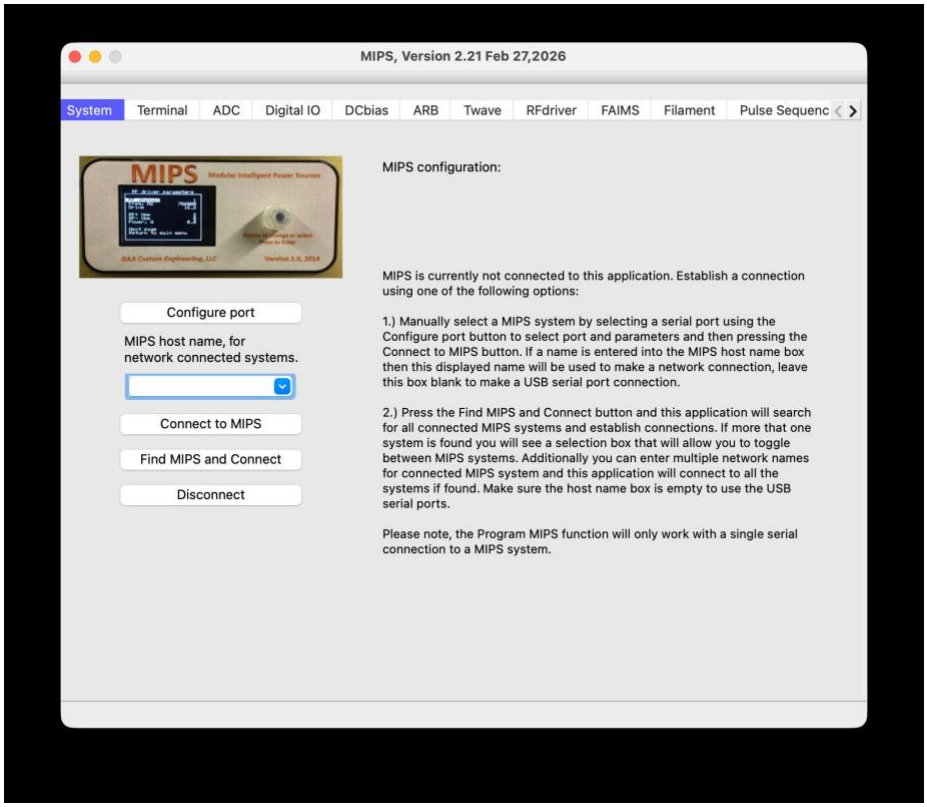

Figure: MIPS application on startup — System tab showing connection controls and instructions

3.1 USB Connection

8. On the System tab, click Configure port.
9. Select the correct COM port and set baud rate to 115,200. Do not change other parameters.
10. Click Apply, then Connect to MIPS.

3.2 Wi-Fi or Ethernet (TCP/IP)

11. Enter the IP address or hostname of the MIPS controller in the MIPS host name field.
12. Click Connect to MIPS.

> **Tip:** Leave the MIPS host name field blank when connecting via USB. If it contains an address the application will prefer TCP/IP.

3.3 Multiple MIPS Systems

- Click Find MIPS and Connect to scan all COM ports and present a selection list.
- Or list multiple IP addresses in the MIPS host name field before connecting.

Each MIPS controller must have a unique name stored in its firmware before connecting multiple units. These names are used in control panel addressing (e.g. MIPS-A, MIPS-B).

4. Main Window Overview

The main MIPS window contains a menu bar, a tab bar, a central content area, and a status bar. Tabs appear dynamically based on detected hardware modules.

Item	Detail
System	Connection management and system-level controls
Terminal	Interactive command-line interface to the MIPS controller
ADC	High-speed analog data acquisition (up to 1 MHz)
Digital IO	Monitor and control digital I/O lines and trigger outputs
DCbias	Set and monitor DC bias voltages across up to 32 channels
RFdriver	Control RF drive frequency, level, and auto-tuning
Twave	Traveling-wave ion guide control and compressor sequencing
ARB	Arbitrary waveform generator for Twave experiments
FAIMS	FAIMS RF and DC control, CV parking and CV scanning
PSG	Pulse sequence generation editor and execution

5. Menu Bar

5.1 File Menu

Load: Opens a file dialog and loads previously saved parameters for the active tab or control panel.

Save: Saves current parameters of the active tab or control panel to a file.

5.2 Terminal Menu

Clear: Clears all text from the Terminal screen.

Message Repeat: Defines a message to be sent to MIPS every second automatically.

Get File from MIPS: Reads a file from the MIPS SD card to the host computer.

Send File to MIPS: Writes a file from the host to the MIPS SD card. Warning: overwrites without confirmation.

Read EEPROM: Saves a module's EEPROM calibration data to a file on the host.

Write EEPROM: Writes EEPROM data from a host file to a selected module.

5.3 Tools Menu

Save Current MIPS Firmware: Reads and saves the current firmware binary from the controller.

Program MIPS: Uploads a new .bin firmware file to the controller.

Select… (Control Panel): Opens a .cfg file to build and display a custom control panel.

Scripting: Opens the scripting console for writing and running ECMAScript automation scripts.

5.4 Help Menu

General Help: Context-sensitive help for the active tab or control panel.

MIPS Commands: Displays the complete MIPS firmware command reference.

5.5 Properties

The Properties dialog is accessed from the About menu. It configures application-level behaviour including startup options, default file paths, and display settings. Settings are saved automatically when you click OK and persist between sessions.

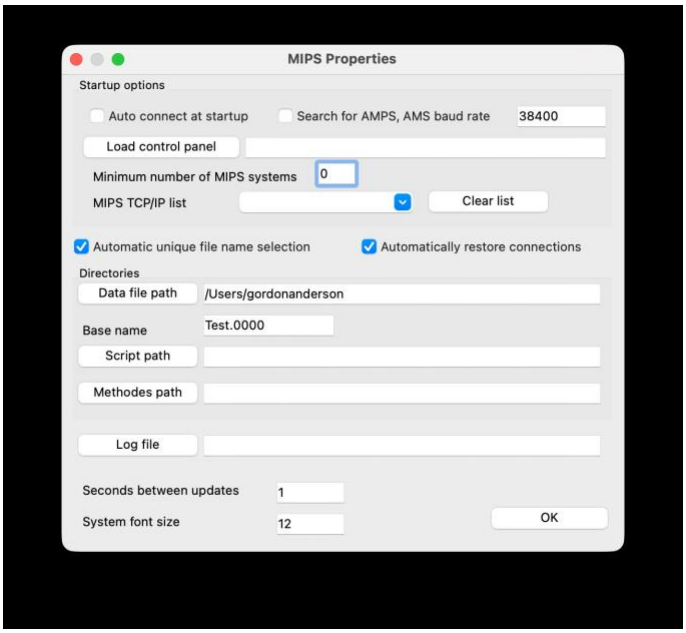

Figure: MIPS Properties dialog

Startup Options

Item	Detail
Auto connect at startup	When checked, MIPS automatically searches for and connects to all available MIPS controllers on launch, without requiring the user to press Connect to MIPS or Find MIPS and Connect.
Search for AMPS, AMS baud rate	When checked, MIPS also searches for AMPS and AMS instruments during auto-connect. Enter the baud rate for these instruments in the adjacent field (default 38400).
Load control panel	Path to a .cfg file to load automatically on startup. Click the button to browse. When set, the control panel opens immediately after connections are established and the main MIPS dialog minimizes.
Minimum number of MIPS systems	Sets the minimum number of MIPS controllers that must be found before the auto-load control panel is opened. Set to 0 to open the panel regardless of how many systems connect. Useful for multi-system setups where the panel requires all units to be present.
MIPS TCP/IP list	A stored list of MIPS network hostnames or IP addresses to connect to. Use the Clear list button to remove all entries. Entries in this list are offered in the MIPS host name dropdown on the System tab.
Automatically restore connections	When checked, MIPS remembers which systems were connected when it was last closed and reconnects to them automatically on the next launch.
Automatic unique file name selection	When checked, data files are saved with an auto-incremented suffix (e.g. Test.0001, Test.0002) to prevent overwriting previous files. When unchecked, the same base name is reused each session.

Directories

Item	Detail
Data file path	Default directory for saving acquired data files. Click the button to browse. The base name field sets the filename stem used when auto-naming is active.
Base name	Filename stem for auto-numbered data files (e.g. Test produces Test.0000, Test.0001, ...).
Script path	Default directory used by the scripting console when opening or saving script files.
Methodes path	Default directory used by Save Method / Load Method dialogs on control panels. (Note: the label in the application uses the French spelling 'Methodes'.)
Log file	Path to an optional log file. When set, MIPS appends timestamped entries whenever connections are made or lost, tabs are switched, control panels are loaded or unloaded, scripts are run, method files are saved or loaded, and data files are acquired.

Display

Item	Detail
Seconds between updates	Controls how frequently the application polls connected MIPS controllers and refreshes widget readbacks. Default is 1 second. Increase for slower systems or decrease for more responsive displays (minimum practical value is 0.1).
System font size	Sets the point size of the application font. Changes take effect after restarting MIPS.

6. Module Tabs

6.1 System Tab

The System tab is always visible and manages all hardware connections. When no MIPS controller is connected the right panel displays connection instructions. Once connected, it displays the current MIPS firmware version and configuration summary.

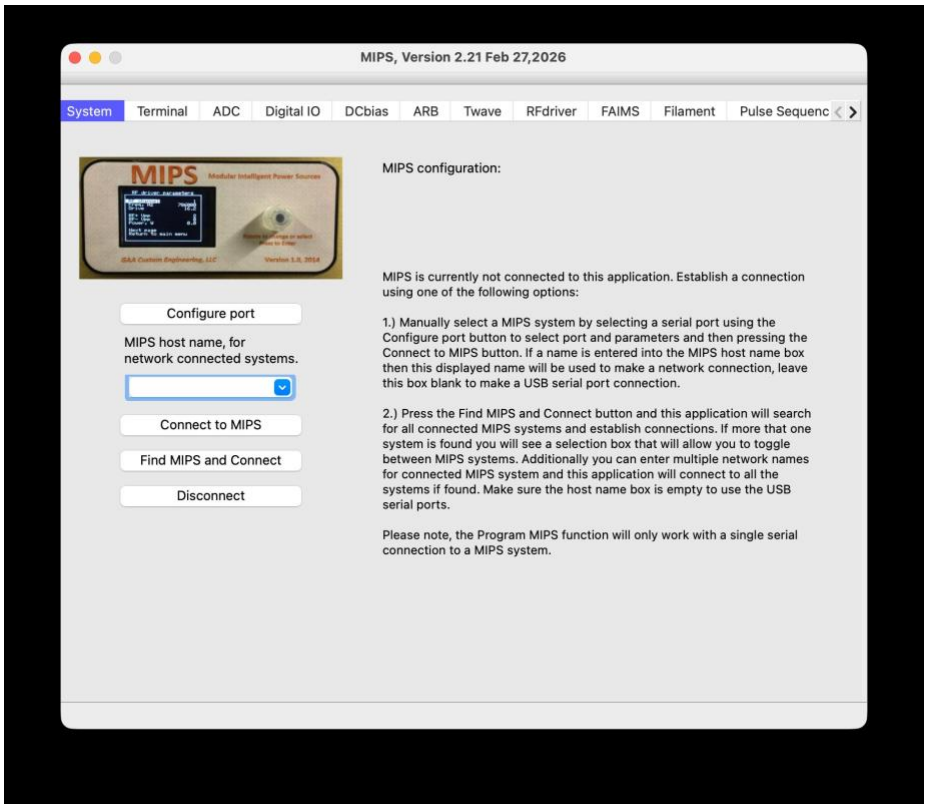

Figure: System tab before connection — showing Configure port, MIPS host name, Connect, Find MIPS and Connect, and Disconnect controls

Item	Detail
Configure port	Opens the serial port configuration dialog. Set the baud rate to 115,200. Leave all other parameters at their defaults. Only needed for USB connections.
MIPS host name	Enter an IP address or hostname for network-connected MIPS systems. Leave blank to use a USB serial port. Multiple hostnames can be entered for multi-system setups.
Connect to MIPS	Establishes a connection using the configured serial port, or via TCP/IP if a hostname is present. Once connected, hardware module tabs appear automatically.

Item	Detail
Find MIPS and Connect	Scans all available COM ports and network addresses to discover MIPS controllers. If multiple systems are found a selection box allows toggling between them.
Disconnect	Closes all active connections and removes the hardware module tabs.

Note: The Program MIPS firmware function (Tools menu) only works with a single USB serial connection. Disconnect any additional MIPS systems before using it.

6.2 Terminal Tab

Provides a direct interactive command interface. Type any MIPS command and press Enter to send it. Responses are displayed in the terminal window. Paste multiple commands to send them in batch.

Important: MIPS commands are case-sensitive. See Help > MIPS Commands for the full reference.

6.3 ADC Tab

Enables high-speed data acquisition from the 0–5 V analog input on the MIPS rear panel.

- Channel / Samples / Vectors / Rate: Configure the acquisition parameters (up to 1 MHz sample rate).
- ADC setup: Downloads configuration to the controller.
- ADC trigger: Initiates an acquisition vector.
- ADC abort: Cancels an ongoing acquisition.

6.4 Digital IO Tab

- Digital Outputs A–P: Sixteen configurable TTL output channels.
- Digital Inputs Q–X: Eight TTL inputs. Q = clock, R = trigger, S–X = general purpose.
- Trigger Output Controls: Set high, low, pulse, or generate a continuous clock with configurable frequency, pulse width, and cycle count.

6.5 DCbias Tab

Controls up to 32 DC bias output channels. Each channel shows a setpoint (editable) and a readback (monitored). The Group field links channels so changing one shifts all members by the same delta.

6.6 RFdriver Tab

Sets RF drive frequency and amplitude for each RF module channel. Auto Tune performs a full frequency scan at 10% drive to find resonance. Auto Retune searches near the current frequency at the current drive level.

6.7 Twave Tab

Controls the Traveling Wave ion guide. Defines an 8-bit pattern that rotates on each clock cycle to create a traveling wave. Configure frequency, pulse voltage (7–100 V), direction, and sync trigger. Also contains the Multi-Pass Compressor Control table (see Section 15).

6.8 ARB Tab

Controls the ARB module in either TWAVE mode or standard ARB mode. Supports the Multi-Pass Compressor Control table (see Section 15) and includes a waveform editor for custom waveform definition.

6.9 FAIMS Tab

Controls the FAIMS separation module. The RF section enables/disables the RF generator and sets drive level. The DC section controls CV and Bias setpoints and readbacks. CV Parking automates CV/Bias changes during an LC separation. Linear and Step scanning modes automate CV sweeps.

6.10 PSG Tab

A graphical editor for defining pulse sequences that change digital outputs and DC bias voltages at precisely timed points. Supports internal or external clock, software or hardware trigger, and nested loops up to 5 levels deep.

The Timing Generator is a control panel widget, not a fixed module tab. It is placed on a custom control panel using the TIMING command — each MIPS system in the panel can have its own Timing Generator instance. See Section 10.9 — Timing Generator in the Widget Reference for full documentation of the editor, events, frame parameters, and scripting API.

6.11 Timing Generator

7. Custom Control Panels — Overview

Custom control panels allow you to build a graphical interface that matches the physical layout and workflow of your specific instrument. A control panel is defined entirely by a plain text configuration file (recommended extension: .cfg) and can display controls from one or more connected MIPS systems on a single screen.

Control panels support:

- An internal tab system for organizing controls across multiple pages
- A rich set of widget types: text fields, buttons, checkboxes, combo boxes, sliders, tables, plots, images, status lights, and timing generators
- Direct binding of widgets to MIPS get/set commands
- Named constants via Define to make panels portable across differently-named systems
- Embedded ECMAScript scripts that run automatically or on user interaction
- A TCP server for remote automation from external applications
- Reusable layout subroutines with parameters, including scrollable GroupBoxes

To load a control panel, first connect to all required MIPS systems, then select Tools > Select… and choose your .cfg file. The main MIPS window will minimize and the control panel will open. Right-click the panel at any time for additional options including help and reload.

> **Note:** Do not use the main MIPS dialog while a control panel is running.

8. Control Panel — Layout Commands

These commands define the overall panel structure and global buttons. They are placed outside any TabSelect block and apply to the whole panel.

Command / Syntax	Description
`# comment`	Any line beginning with # is a comment and is ignored.
`size,x,y`	Sets the width and height of the control panel in pixels. Should be the first command in the file.
`image,filename`	Sets a background image (full path required). The image is stretched to fill the panel size.
`Help,filename`	Specifies a text file containing help content for this panel. Accessible by right-clicking the panel.
`Define,name,value`	Defines a named constant. Wherever 'name' appears as the MIPSname argument in any Ccontrol or hardware widget command, it is substituted with 'value' at parse time. This makes panels portable across systems with different names.
`TextLabel,text,font,x,y`	Places a static text label at position (x,y) using the specified font size in points.
`Shutdown,name,x,y`	Adds a Shutdown button. Pressing it zeros all RF drive levels and disables all DC bias outputs.
`SaveLoad,save,load,x,y`	Adds Save and Load buttons for saving and restoring all panel settings to/from a file.
`MIPScomms,x,y`	Adds a button that opens a direct MIPS command dialog.
`Script,x,y`	Adds a Scripting button that opens the scripting console.
`CPbutton,name,file,x,y`	Adds a button that reloads the panel using the specified cfg file.
`TCPserver,port`	Opens a TCP server on the specified port. External applications can connect and send mips.Command()-style strings to remotely control the panel.
`SkipCount,n`	Sets the number of polling cycles to skip between automatic UI updates. Higher values reduce communication load.
`SendCommand,MIPSname,cmd[,arg]`	Sends a MIPS command to the named system once at panel load time. Used for hardware initialization. Multiple SendCommand lines are executed in order.
`CallOnStart,TRUE`	Causes the ScriptButton defined immediately before this command to be automatically pressed when the panel loads.
`InitParms,filename.ini`	Runs MIPS commands from the specified file at panel load time. Each line must be formatted as: MIPSname,command.

9. Control Panel — Tab System

Control panels support an internal tab bar that organizes controls across multiple named pages, independent of the main MIPS application tab bar.

Command / Syntax	Description
`Tab,name,x,y,w,h,Tab1,Tab2,...`	Creates a tab widget named 'name' at position (x,y) with width w and height h. The remaining arguments are the names of the tabs.
`TabSelect,tabwidget,tabname`	Begins a block of commands placed on the named tab. All widget commands that follow apply to this tab until TabSelectEnd.
`TabSelectEnd`	Ends the current TabSelect block.

Example: Tab,Tab,700,400,200,45,DCB RF,Scripts,Plot creates a tab widget with three tabs. Contents are defined in separate TabSelect/TabSelectEnd blocks.

Controls inside a TabSelect block use coordinates relative to the tab content area. Controls defined outside any TabSelect block appear on the main panel background. Multiple Tab widgets can be defined on the same panel.

10. Control Panel — Widget Reference

The following widgets can be placed on a control panel or within a tab. All positions (x, y) are in pixels relative to the containing area.

10.1 Hardware Channel Widgets

These widgets bind directly to specific MIPS hardware module channels and automatically handle polling and communication with the controller.

Command / Syntax	Description
`DCBchannel,name,MIPSname,chan,x,y`	DC bias channel with a setpoint field (left, editable) and a readback field (right, read-only). name is the display label, MIPSname is the MIPS system name, chan is the channel number (1–32).
`DCBoffset,name,MIPSname,board,x,y`	DC bias offset control for the specified module board number.
`DCBenable,name,MIPSname,x,y`	Checkbox to enable or disable the DC bias power supply.
`RFCchannel,name,MIPSname,chan,x,y`	RF channel control widget with full closed-loop options: drive level, setpoint, frequency, RF+/RF- voltages, power, open/closed loop selection, and Tune/Retune buttons.
`RFchannel,name,MIPSname,chan,x,y`	Simplified RF channel widget showing Drive, Freq, RF+, RF-, Power, Tune and Retune. Does not include closed-loop control options.
`DIOchannel,name,MIPSname,chan,x,y`	Digital I/O channel widget for TTL input/output control.
`ESIchannel,name,MIPSname,chan,x,y`	ESI (electrospray ionization) voltage channel control widget.
`ARBchannel,name,MIPSname,chan,x,y`	ARB channel control widget showing frequency, amplitude, aux output, offset output, waveform type selector, waveform editor button, and Forward/Reverse controls.
`RFAMP,name,MIPSname,chan,x,y`	RF amplifier widget. Renders an internal tabbed control with sub-tabs: RF settings (enable, open/closed loop, frequency, drive, setpoint, RF+/RF- voltages, power), Mass filter (m/z, Ro, k, resolution, Update button), and RF amp. Sub-tab controls are addressable via mips.Command() as: TabName.WidgetName.SubTabName.ControlName.
`COMPRESSOR,name,MIPSname,x,y`	Adds a Compressor control button linked to the ARB/Twave compressor functionality on the named MIPS system.
`TIMING,name,MIPSname,x,y`	Timing Generator widget. Embeds a full pulse sequence generator with an interactive editor dialog. Handles arbitrarily complex event-based timing sequences. Multiple TIMING widgets can appear on the same panel.

Command / Syntax	Description
	The name is used for mips.Command() addressing. See Section 6 for full documentation.
`EventControl,name,event,x,y`	Reads and writes a named event parameter in the most recently defined TIMING widget. For example, gate.Value reads/writes the gate event on-value.

10.2 Ccontrol — Generic Control Widget

Ccontrol is the most versatile widget in the cfg system. It serves two distinct and equally important roles that can be used independently or together.

Hardware-bound mode: when a real MIPS system name and MIPS commands are provided, the control communicates directly with hardware. The widget polls the hardware on every update cycle using the get command and displays the result; when the user changes the value the set command is sent immediately. This makes it possible to surface any MIPS command as a first-class UI element without writing any script code.

UI-only (scripting) mode: when the MIPSname field does not match any connected MIPS system, the control has no hardware binding. It acts purely as a UI element — a text field, checkbox, button, or drop-down that scripts can read from and write to freely. This is how panels capture user input (parameters, timing values, mode selections) and present script-calculated results back to the operator.

The two modes are selected entirely by what is in the MIPSname field. The rest of the syntax is identical.

Syntax:

```
Ccontrol,label,MIPSname,type,getCmd,setCmd,readbackCmd,units,x,y[,modifier]
```

Item	Detail
label	The display name shown next to the control. Also used as the object name in mips.Command() addressing.
MIPSname	The MIPS system to communicate with. Must exactly match a connected system name for hardware binding. Any other value (Enter, Pulse, Parameter, Number, Scan, Slider, Voltage, etc.) makes the control UI-only with no hardware polling. A Define name is substituted here at parse time, enabling portable panels.
type	Widget type: LineEdit, Button, CheckBox, or ComboBox.
getCmd	Hardware mode: MIPS command sent to read the current value (e.g. GADCRATE). UI-only mode: leave blank; the field retains whatever value was last written to it by a script or the user.

Item	Detail
setCmd	Hardware mode: MIPS command sent when the value changes (e.g. SADCRATE). UI-only mode: can be left blank, or set to a descriptive label that is returned by ObjectName.setCmd and used as a tooltip.
readbackCmd	Hardware mode: MIPS command for a separate read-only readback field shown alongside the setpoint. For CheckBox, append _TRUE_FALSE or _LOW_HIGH to map the hardware response to checked/unchecked state. Leave blank if not needed.
units	Units label shown to the right of the control (e.g. Hz, V, mS). Leave blank if not needed.
x, y	Position in pixels within the containing area.
modifier	Optional: STRING (plain text, no numeric validation), STRINGL (wide text field for longer strings).

Hardware-Bound Examples

These controls talk directly to a connected MIPS system. The MIPSname matches a real system name and MIPS commands are provided:

```
// LineEdit bound to ADC rate — polls GADCRATE, sends SADCRATE on change
Ccontrol,Rate,MIPS,LineEdit,GADCRATE,SADCRATE,,Hz,10,55

// Read-only readback — only readback command supplied, no setpoint field
Ccontrol,Recorded,MIPS,LineEdit,,,ADCRBVECS,,10,130

// Button that sends a MIPS command when pressed
Ccontrol,Trigger,MIPS,Button,,ADCRBTRIG,,,10,180

// CheckBox mapped to hardware ON/OFF state
Ccontrol,Enable,MIPS-
A,CheckBox,SENA_FALSE,SENA_TRUE,GENA_TRUE_FALSE,,10,20

// ComboBox bound to hardware mode selection
Ccontrol,Mode,MIPS,ComboBox,GSTYP,SSTYP,,,20,30
ComboBoxList,FREQ,MZ,T1,T3
```

UI-Only (Scripting) Examples

These controls have no hardware binding. Scripts read and write them to exchange data with the operator:

```
// Parameter inputs — user types values, scripts read them
Ccontrol,Period,Pulse,LineEdit,,period in mS,,mS,20,25
Ccontrol,Cycles,Number,LineEdit,,of cycles,,,20,50

// Result display — script writes calculated value, user sees it
Ccontrol,A+B,Parameter,LineEdit,,A+B,,,20,130
```

```
// Mode selector with fixed options, no hardware command
Ccontrol,Type,Signal,ComboBox,,type Pulse or Ramp,,,390,30
ComboBoxList,Pulse,Ramp

// Checkbox flag read by script
Ccontrol,Ext trig,Enable,CheckBox,,external trigger on R input,,,20,75
```

CheckBox Response Mapping

For hardware-bound CheckBox controls, the readback command field encodes the hardware responses that correspond to the checked and unchecked states. The format is COMMAND_CHECKED_UNCHECKED:

- GENA_TRUE_FALSE — send GENA; response TRUE = checked, FALSE = unchecked
- GRFAGAIN_1_LOW_HIGH — send GRFAGAIN,1; response LOW = unchecked, HIGH = checked
- GSWAIT_TRUE_FALSE — send GSWAIT; response TRUE = checked, FALSE = unchecked

Off-Canvas Storage Variables

A Ccontrol placed at a negative position (e.g. x=-100, y=-100) is invisible to the user but fully accessible to scripts. This is the standard pattern for storing values that scripts need to share across function calls, or for holding a resolved MIPS system name when using Define:

```
Define,name,MIPS-A
Ccontrol,Aname,,LineEdit,,name,,,-100,-100
```

The hidden control stores the string name. Scripts read it back with:

```
var sysName = mips.Command("Aname").trim()
mips.SendCommand(sysName, "GDCB,1\n")
```

> **Key rule:** Whether a Ccontrol is hardware-bound or UI-only is determined solely by the MIPSname field. If MIPSname matches a connected system, the control polls hardware automatically. If it does not match, there is no hardware polling and the control value is entirely under script control.

10.3 GroupBox

Groups controls inside a labeled border box. Standard form creates a fixed-size box; extended form creates a scrollable box.

Standard (fixed size):

```
GroupBox,title,width,height,x,y
  ... widget commands ...
GroupBoxEnd
```

Scrollable (visible window with larger scrollable canvas):

```
GroupBox,title,visibleWidth,visibleHeight,x,y,canvasWidth,canvasHeight
  ... widget commands ...
GroupBoxEnd
```

The first width/height defines the visible window size. The optional canvasWidth/canvasHeight defines the full scrollable content area inside. Scrollbars appear automatically when the content is larger than the visible area.

Controls inside a GroupBox are addressed in scripts as: TabName.GroupBoxTitle.ControlName

10.4 Slider

```
Slider,name,orientation,min,max,default,x,y
```

Item	Detail
name	Control name for mips.Command() addressing.
orientation	H for horizontal, V for vertical.
min / max / default	Range and initial value.
x, y	Position in pixels.

10.5 Table

```
Table,name,rowHeight,numCols,colWidth,numRows,x,y
```

Table cells are read and written from scripts:

- Get row count: mips.Command("Tab.TableName.Rows")
- Get column count: mips.Command("Tab.TableName.Columns")
- Read a cell: mips.Command("Tab.TableName.Cell,row,col")
- Set a column header: mips.Command("Tab.TableName.Header,col=Title")

10.6 ImageBox

```
ImageBox,filename,width,height,x,y
```

10.7 StatusLight

```
StatusLight,name,x,y
```

10.8 PlotData

Defines an embedded plot widget. Scripts use mips.Command() to send plot commands to it using the syntax: mips.Command("Tab.PlotName=PlotCommand")

```
PlotData,name,yAxisLabel,xAxisLabel,numGraphs,width,height,x,y
```

10.9 Timing Generator

The Timing Generator provides a graphical pulse sequence editor that programs precisely timed digital output events on the MIPS controller. It is designed for applications such as ion injection, trapping, and release sequences in mass spectrometry, where many output signals must fire at exact clock-cycle-accurate times relative to each other.

Add the Timing Generator to a custom control panel cfg file with:

```
TIMING,name,MIPSname,x,y
```

Where name is the label shown on the panel button, MIPSname is the connected MIPS system, and x,y is the button position. This adds three buttons to the panel: Edit (opens the sequence editor), Trigger (starts a run), and Abort (stops an active run).

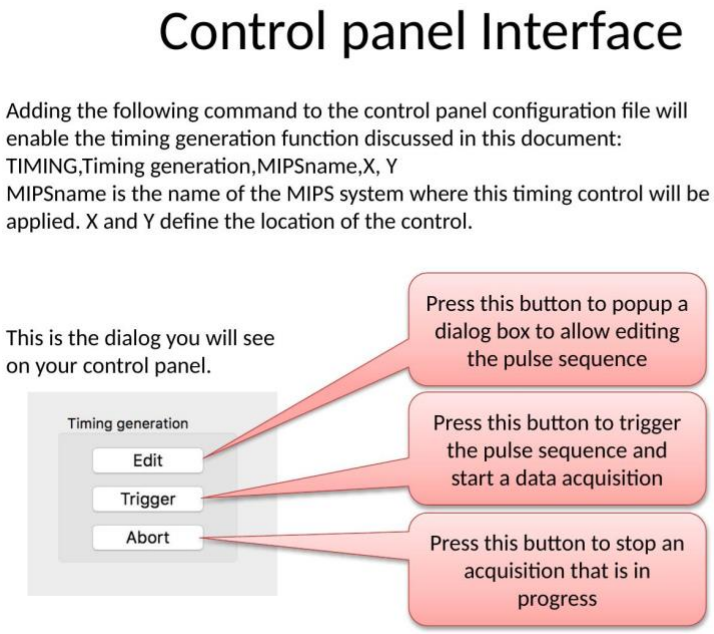

Figure: Timing Generator control panel widget showing Edit, Trigger, and Abort buttons

10.9.1 Pulse Sequence Editor

Pressing the Edit button opens the Timing generation editor dialog. This dialog is divided into three main areas: the Event editor (left), the Frame parameters (center), and the Clock and trigger options (right).

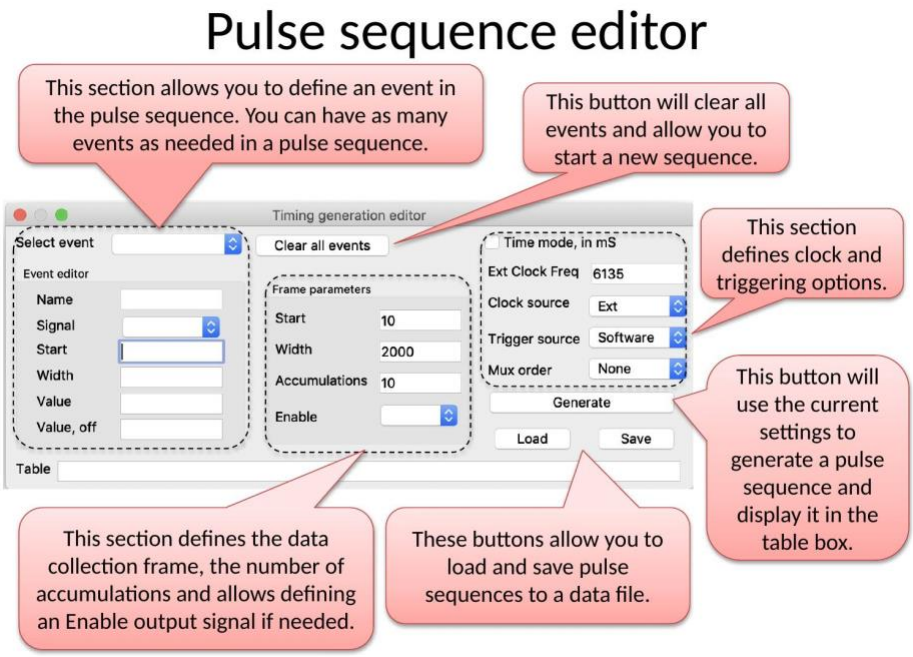

Figure: Timing generation editor dialog — full overview with all three sections

10.9.2 Events

An event defines a single named output signal change within the sequence. Each event has a unique name, a signal to control, a start time, a width (duration), and an active and inactive output value. You can define as many events as needed. Multiple events can share the same start and width values, but every event name must be unique.

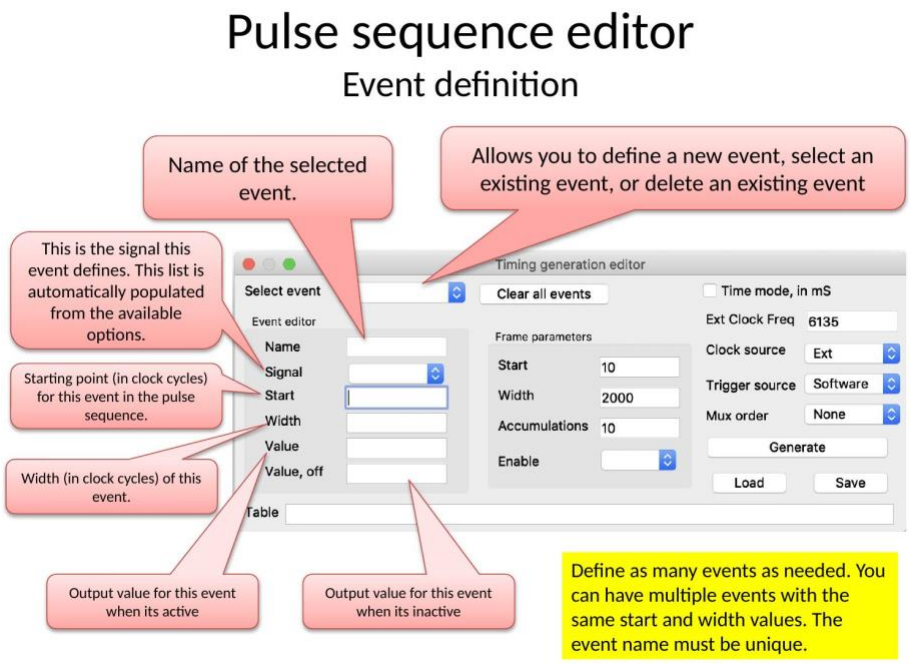

Figure: Event definition section — Name, Signal, Start, Width, Value, and Value,off fields

Item	Detail
Select event	Spinner/dropdown to select an existing event, create a new one, or delete the currently selected one.
Name	Unique name for this event. Used to reference it in expressions in other events.
Signal	The output signal this event controls. The list is automatically populated from the available DIO outputs on the connected MIPS system.
Start	Start time for this event in clock cycles (or milliseconds if Time mode is enabled). Can be a fixed number or an expression referencing other events (e.g. Inject time.Start).
Width	Duration of the active pulse in clock cycles (or milliseconds). Can be a fixed number or an expression.
Value	Output state when this event is active.
Value, off	Output state when this event is inactive.

Linked events: Start and Width fields accept arithmetic expressions that reference other event values using the syntax EventName.Start or EventName.Width. Both + and − are supported. This lets you chain events so that adjusting one automatically repositions all the dependent events. For example: Trap time.Start - Fill time.Width sets the Fill time start relative to the Trap time.

10.9.3 Frame Parameters

The Frame parameters define the overall envelope of the pulse sequence and the data acquisition window within it.

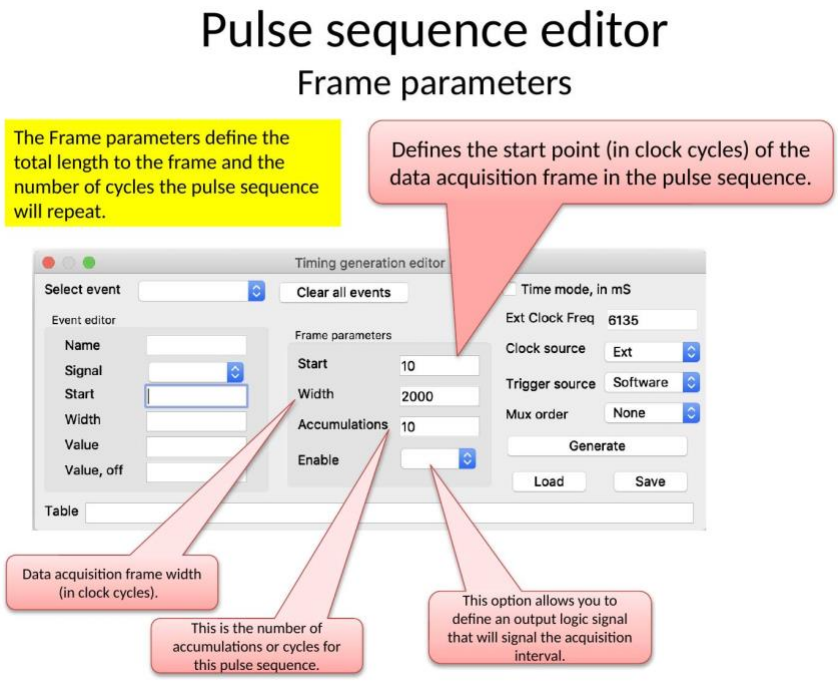

Figure: Frame parameters — Start, Width, Accumulations, and Enable fields

Item	Detail
Start	Start point of the data acquisition frame within the sequence, in clock cycles.
Width	Width of the data acquisition frame (total active collection period), in clock cycles.
Accumulations	Number of times the full pulse sequence repeats per triggered run.
Enable	Optional DIO output signal to assert during the acquisition frame interval, signalling external instruments that data collection is active.

10.9.4 Clock and Trigger Options

The right panel of the editor controls the timing source and the trigger that starts a run.

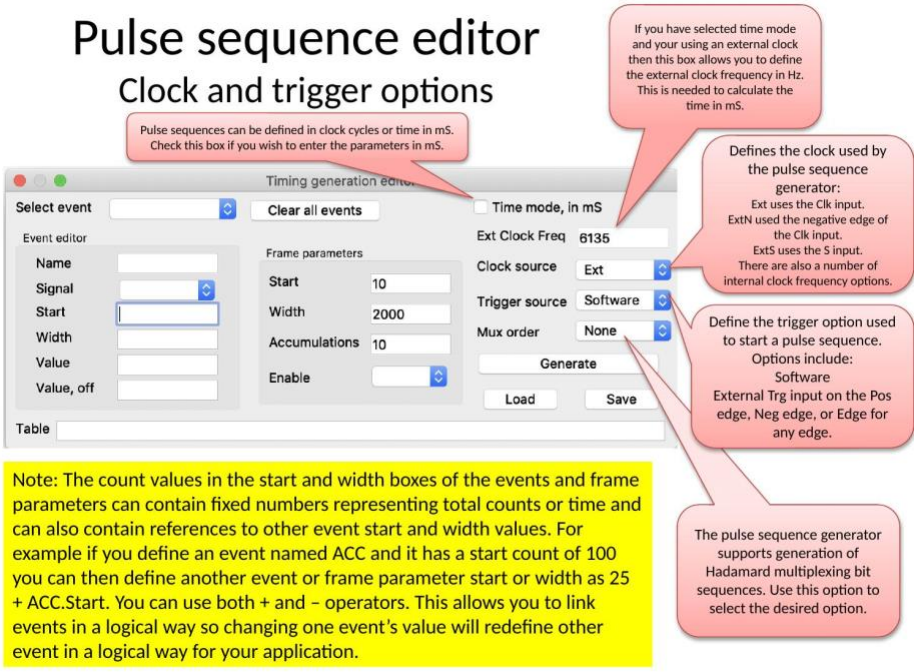

Figure: Clock source, Trigger source, Mux order, Time mode, and external clock frequency

Item	Detail
Time mode, in mS	When checked, all Start and Width values are entered in milliseconds rather than raw clock cycles. The external clock frequency field must be set correctly for the conversion to be accurate.
Ext Clock Freq	The frequency in Hz of the external clock. Only used when Time mode is enabled and an external clock source is selected.
Clock source	Selects the clock driving the sequence generator. Ext uses the CLK input (rising edge). ExtN uses the CLK input (falling edge). ExtS uses the S input. Internal options include 42000000, 10500000, 2625000, and 656250 Hz.
Trigger source	Defines how a run is started. Software triggers immediately when the Trigger button is pressed. External options trigger on the Pos edge, Neg edge, or any Edge of the external trigger input.
Mux order	Selects a Hadamard multiplexing bit sequence order. Set to None for standard (non-multiplexed) operation. See Section 10.9.6 for details.

10.9.5 Example Pulse Sequence

This example shows a four-event ion trapping sequence where all events are linked so that only the Inject time Start needs to be changed to reposition the entire sequence. The events are:

- Fill time — width 100 cycles; ions fill the trap
- Trap time — width 10 cycles; ions are held while the gate closes
- Release time — width 20 cycles; ions are released toward the detector
- Inject time — start 200, width 9 cycles; the injection gate pulse

Item	Detail
Fill time	Start: Trap time.Start − Fill time.Width | Width: 100
Trap time	Start: Inject time.Start − Trap time.Width | Width: 10
Release time	Start: Inject time.Start | Width: 20
Inject time	Start: 200 | Width: 9

With this arrangement, changing Inject time.Start to any value automatically repositions Fill time and Trap time relative to it. Signals controlled by each event depend on the instrument wiring. Trap time in this example controls no signal — it is a pure delay representing the trapping interval.

10.9.6 Hadamard Multiplexing

The Timing Generator supports automatic generation of Hadamard multiplexing injection sequences. Hadamard multiplexing improves signal-to-noise by encoding many injections into a single sequence according to a binary Hadamard matrix row, then decoding the accumulated result mathematically.

To use Hadamard multiplexing, define the standard four-event sequence (Fill time, Trap time, Release time, Inject time) exactly as described in the example above. The Inject time event's Start must be set to zero. Set the Mux order drop-down to the desired Hadamard order. When Generate is pressed, the system calculates all injection start times from the bit pattern and generates the full multiplexed sequence automatically. Events with any other names in the sequence are used as defined and control other aspects of the experiment independently.

10.9.7 Generate, Load, and Save

Pressing Generate compiles all events and frame parameters into a pulse table and downloads it to the MIPS controller. The compiled table is displayed in the Table area at the bottom of the editor for inspection. Load and Save allow pulse sequences to be stored as data files and recalled between sessions.

11. Control Panel — Subroutines

Subroutines define reusable blocks of widget layout commands with named parameters, allowing the same layout to be instantiated multiple times with different arguments.

```
Subroutine,SubroutineName,Param1,Param2,...
  ... widget commands using Param1, Param2 as placeholders ...
EndSubroutine

Call,SubroutineName,Value1,Value2,...,x,y
```

When Call is executed, the subroutine body is instantiated with each parameter replaced by its corresponding value. The final x,y in the Call command sets the position of the instantiated block.

Subroutines can be nested — a subroutine can contain Call commands that invoke other subroutines. Subroutines can also be called from within another Subroutine definition, enabling composite layouts.

> **Example:** A ChannelDialog subroutine defines a GroupBox containing a PlotData widget and a Table. Calling it three times with different names creates three identical channel editor groups stacked vertically inside a scrollable GroupBox.

12. Control Panel — Scripting System

Control panel cfg files support embedded ECMAScript (JavaScript) scripts. Scripts can respond to user interactions, automate sequences, read and write control values, communicate with MIPS hardware, and generate plots.

12.1 Defining Scripts

```
Script,ScriptName
  // JavaScript code here
EndScript
```

12.2 ScriptObj — Script Aliases

ScriptObj creates a named alias associating a script object name with a script name. Used to register scripts as event handlers:

```
ScriptObj,ObjectName,ScriptName
```

12.3 ScriptButton

Adds a clickable button that executes a named script:

```
ScriptButton,label,ScriptName,x,y
```

To call a script with parameters, use the call expression syntax:

```
ScriptButton,label,ScriptName,"call(arg1,arg2)",x,y
```

The script must be defined as a JavaScript function expression to accept parameters:

```
Script,ScriptName
(
  function(a, b) {
    mips.popupMessage("a=" + a + ", b=" + b)
  }
)
EndScript
```

Pressing a ScriptButton while its script is already running will abort the running script. This makes ScriptButtons naturally act as start/abort toggles for long-running operations.

12.4 Script,StartUp — Automatic Initialization

A script named StartUp is automatically executed once when the control panel finishes loading. Use it to:

- Set column headers on Table widgets
- Set initial default values
- Attach event handler scripts to controls using .script= and .scriptCall=
- Set initial control colors
- Enable asynchronous messaging
- Load a settings file

```
Script,StartUp
  mips.Command("Tab.Table.Header,0=Time")
  mips.Command("Tab.Control.script=MyScript")
  mips.Command("Tab.Control.color=yellow")
  mips.Command("Load,defaults.settings")
EndScript
```

12.5 Script,Common — Shared Utility Functions

A script named Common is automatically appended to every other script in the system when executed. Define utility functions here to make them available everywhere without explicit imports:

```
Script,Common
  function getInt(a) { return parseInt(mips.Command(a).trim()) }
  function getFloat(a) { return parseFloat(mips.Command(a).trim()) }
  function getBool(a) { return mips.Command(a).trim() == "TRUE" }
  function getName() { return mips.Command("Aname.setCmd").trim() }
EndScript
```

A common pattern is to define a getPath() function in Common that returns the GroupBox path prefix for the panel, so all scripts can reference controls without hard-coding the full path:

```
function getPath() { return "TIMS TOF timing." }
```

12.6 Attaching Scripts to Control Events

Attach scripts to control change events from within StartUp:

```
mips.Command("TabName.ControlName.script=ScriptName")
```

To pass parameters when the event fires:

```
mips.Command("TabName.ControlName.script=ScriptName")
mips.Command("TabName.ControlName.scriptCall=call(arg)")
```

Example: mips.Command("MIPS.DriftTime.script=timingChanges") fires timingChanges every time the DriftTime slider value changes.

12.7 CallOnStart

Placing CallOnStart,TRUE immediately after a ScriptButton causes that button to be automatically pressed when the panel loads. This is useful for running an initialization script or starting a background monitoring loop at startup:

```
ScriptButton,Set default,Default.scrpt,20,200
CallOnStart,TRUE
```

13. mips Object API Reference

The mips object is exposed to all scripts in the control panel scripting system. It provides functions for reading and writing UI controls, communicating with MIPS hardware, managing plots, displaying messages, and controlling system state.

13.1 mips.Command() — UI Control Access

mips.Command() is the primary mechanism for interacting with control panel widgets. It uses a dot-path addressing scheme to identify controls.

Addressing Syntax

- Flat control on a tab: TabName.ControlName
- Control inside a GroupBox: TabName.GroupBoxTitle.ControlName
- Control inside RFAMP sub-tab: TabName.RFAMPName.SubTabName.ControlName
- Timing generator event: TimingName.Event.PropertyName
- Table property: TabName.TableName.Rows / .Columns / .Cell,r,c
- Panel-level control (no tab): ControlName (e.g. a GroupBox directly on the panel)

Get a value

```
value = mips.Command("TabName.ControlName").trim()
```

Set a value

```
mips.Command("TabName.ControlName=newValue")
```

Control Properties

Command / Syntax	Description
`Tab.Control.script=Name`	Attaches the named script to fire when this control's value changes.
`Tab.Control.scriptCall=call(args)`	Defines the call expression used when the attached script fires, enabling parameters to be passed.
`Tab.Control.color=colorname`	Sets the background color of the control (CSS color names: yellow, pink, transparent, black, etc.).
`Tab.Control.hide=TRUE/FALSE`	Hides or shows the control.
`Tab.Control.toolTip=text`	Sets the tooltip text for the control.
`Tab.Control.getCmd=CMD`	Dynamically sets the MIPS get command for the control.
`Tab.Control.setCmd=CMD`	Dynamically sets the MIPS set command for the control.

Command / Syntax	Description
`Tab.Control.channel`	Returns the hardware channel number associated with a hardware widget (e.g. DCBchannel).
`Tab.Table.Header,n=Title`	Sets the header text for column n of a Table widget.
`Tab.Table.Rows`	Returns the number of rows in a Table widget.
`Tab.Table.Columns`	Returns the number of columns in a Table widget.
`Tab.Table.Cell,r,c`	Returns the value of the cell at row r, column c.
`TGName.Event.Select event=Name`	Selects the named event in a Timing Generator for subsequent property access.
`TGName.Event.Width`	Gets/sets the width (duration) of the currently selected timing event.
`TGName.Frame.Width`	Gets/sets the overall frame width of the timing generator.
`TGName.Abort`	Stops the timing generator.
`TGName.Generate`	Downloads the timing table to MIPS.
`TGName.Trigger`	Triggers the timing generator to start.
`MIPSname.enableAsync=TRUE`	Enables asynchronous message mode on the named MIPS system, allowing MIPS to push unsolicited status messages to the application.
`MIPSname.getMessage`	Polls the async message queue for the named MIPS system. Returns the next pending message string, or empty string if none.
`Load,filename`	Loads a settings file. File is assumed to be in the same folder as the cfg file.
`Tab.PlotName=PlotCommand`	Sends a plot command to an embedded PlotData widget (see Section 13.4).

13.2 Hardware Communication Functions

Function	Description / Return Value
`mips.SendCommand(name, cmd)`	Sends cmd to the MIPS system named name. Returns true if accepted, false if rejected. Always terminate commands with \n.
`mips.SendMess(name, cmd)`	Sends cmd to the named MIPS system and returns the response string.
`mips.SendCommand(cmd)`	Sends cmd to the currently active MIPS system (single-system context).
`mips.SendMess(cmd)`	Sends cmd to the currently active MIPS system and returns the response.

13.3 User Interaction Functions

Function	Description / Return Value
`mips.msDelay(ms)`	Pauses script execution for ms milliseconds. Must be used in loops to prevent UI lockup.
`mips.statusMessage(msg)`	Displays msg on the application status bar.
`mips.popupMessage(msg)`	Displays a modal popup with an OK button. Pauses until dismissed.
`mips.popupYesNoMessage(msg)`	Displays a Yes/No dialog. Returns true if Yes, false if No.
`mips.popupUserInput(title, msg)`	Displays a text input dialog. Returns the user-entered string.
`mips.SelectFile(mode, title, ext)`	Opens a file selection dialog. mode is 'Open' or 'Save', ext is the file extension filter. Returns the selected file path string.

13.4 Plotting Functions

The plotting API creates and populates dynamic data plots. PlotData widgets in the cfg provide embedded plot areas; mips.CreatePlot() can create standalone plot windows.

Function	Description / Return Value
`mips.CreatePlot(title, yLabel, xLabel, n)`	Creates a new standalone plot window with n graphs.
`mips.PlotCommand(cmd)`	Sends a command to the most recently created standalone plot.

For embedded PlotData widgets, use mips.Command() instead:

```
mips.Command("Tab.PlotName=PlotCommand")
```

Common plot command strings:

Command / Syntax	Description
`NewGraph,graphNum,points`	Initializes graph number graphNum with a data buffer of points points.
`Xrange,min,max,labelMin,labelMax`	Sets the X axis display range and label range.
`Xrange,Time`	Sets X axis to show elapsed time.
`Yrange,min,max`	Sets the Y axis display range.
`Yrange,Auto`	Auto-scales the Y axis.
`PlotPoint,x,y`	Adds a single data point.
`AddPoint,i,x,y`	Adds a point at index i with coordinates (x,y).
`AddPoints,graph,start,data`	Adds a comma-separated list of values starting at index start on the specified graph.
`Plot`	Renders all pending data.
`Refresh`	Refreshes the plot display.

Command / Syntax	Description
`Plot1,name`	Names the first graph series.
`Title,text`	Sets the plot title.
`NormalCursor`	Sets the cursor to normal mode.
`ClearComments`	Clears all comment annotations.
`PlotOnly,TRUE/FALSE`	Enables or disables plot-only rendering mode.
`PLOTONLY,TRUE/FALSE`	Alternate casing form of PlotOnly.
`Average`	Averages all current graph scans and displays the result.
`CreatePlot,name,y,x,n`	Creates a named sub-plot within a PlotData widget.
`Load,filename`	Loads a previously saved plot file into the plot window.
`#text`	Writes a comment/metadata annotation line into the plot data file. Useful for embedding scan parameters.

13.5 Update Control Functions

Function	Description / Return Value
`mips.UpdateHalted(bool)`	Pass true to stop all background UI polling; pass false to resume. Use before long hardware communication sequences to prevent interference from the update timer.
`mips.sysUpdating()`	Returns true if a background update cycle is currently in progress. Use after mips.UpdateHalted(true) to confirm the system has fully stopped before proceeding.

Pattern: The recommended pattern before a scan: call mips.UpdateHalted(true), then loop on mips.sysUpdating() until it returns false, then perform hardware communication, then call mips.UpdateHalted(false) when done.

13.6 Control Panel Management Functions

Function	Description / Return Value
`mips.Save(filename)`	Saves all control panel settings to the specified full file path.
`mips.Load(filename)`	Loads settings from the specified full file path and applies them to the panel.
`mips.SystemShutdown()`	Safe shutdown: disables DC bias, zeros RF drive levels and ESI voltages. Saves settings first.
`mips.SystemEnable()`	Restores the system from a shutdown state.
`mips.Acquire(filepath)`	Starts a data acquisition. Non-blocking. Returns true if started successfully.
`mips.isAcquiring()`	Returns true if an Acquire operation is currently running.

Function	Description / Return Value
`mips.DismissAcquire()`	Closes the Acquire console dialog after acquisition completes.

Caution: Never write a tight loop without a mips.msDelay() call inside it. A loop with no blocking call will prevent UI event processing and lock the application.

13.7 ZMQ Messaging Interface

The mips object exposes a ZMQ (ZeroMQ) request-reply interface that allows control panel scripts to communicate with external processes over a TCP socket using the ZMQ REQ-REP pattern. This is useful for integrating MIPS with data acquisition software, lab automation systems, or any other process that can speak ZeroMQ.

All ZMQ operations are performed through a single function:

```
result = mips.ZMQ(cmd)   // where cmd is one of the strings below
```

Function	Description / Return Value
`mips.ZMQ("BEGIN,address")`	Opens a ZMQ REQ socket and connects to the specified TCP address. Example address: tcp://localhost:5555. Returns empty string on success, '?' if already connected.
`mips.ZMQ("REQUEST,message")`	Sends message to the connected server asynchronously. The reply is stored internally and retrieved with REPLY. Returns empty string immediately — does not block.
`mips.ZMQ("REPLY")`	Returns the reply string received from the last REQUEST. If no reply has arrived yet, returns an empty string. Call after a mips.msDelay() to give the server time to respond.
`mips.ZMQ("ERROR")`	Returns the error string from the last failed REQUEST, or empty string if no error occurred.
`mips.ZMQ("STOP")`	Closes the ZMQ socket and stops the worker thread.

Installation: ZMQ support requires the ZeroMQ and cppzmq libraries to be installed on the host computer. On macOS use Homebrew: brew install zmq cppzmq. On Windows install cppzmq via vcpkg. The MIPS application must be compiled with these libraries linked.

Typical usage pattern in a script:

```
// Connect to external server
mips.ZMQ("BEGIN,tcp://localhost:5555")

// Send a request
mips.ZMQ("REQUEST,Hello server")

// Wait for the server to respond
```

```
mips.msDelay(100)

// Read the reply
var reply = mips.ZMQ("REPLY")
mips.statusMessage(reply)

// Check for errors
var err = mips.ZMQ("ERROR")
if(err != "") mips.popupMessage("ZMQ error: " + err)

// Disconnect when done
mips.ZMQ("STOP")
```

Note: REQUEST is non-blocking — it fires the message and returns immediately. Always call mips.msDelay() before reading REPLY to allow the server time to respond. The default socket timeout is 1000 ms; if no reply is received within that window, the error is stored and retrievable via ERROR.

13.8 Context Availability Summary

Item	Detail
mips.Command()	All control panel contexts
mips.SendCommand(name,cmd) / mips.SendMess(name,cmd)	All contexts (multi-system form)
mips.SendCommand(cmd) / mips.SendMess(cmd)	Single-system context only (Tools > Scripting)
mips.msDelay(), mips.statusMessage(), popup functions	All contexts
mips.SelectFile()	All contexts
mips.CreatePlot(), mips.PlotCommand()	All contexts
mips.UpdateHalted(), mips.sysUpdating()	All contexts
mips.Save(), mips.Load(), mips.SystemShutdown(), mips.SystemEnable()	Control panel context only
mips.Acquire(), mips.isAcquiring(), mips.DismissAcquire()	Control panel with IFT defined
mips.ZMQ()	All contexts

14. Object Script Reference

Every object placed on a control panel — DCBchannel, RFchannel, Table, Slider, and all others — exposes a set of scriptable properties. Scripts read and write these properties using mips.Command() with the object's full address path. Each object type supports different properties, all derived directly from the source implementation.

14.0 Object Addressing Rules

The address used in mips.Command() depends on where the object is placed in the panel hierarchy. The rules are:

Location	Address to use in mips.Command()
Object on the panel (no container)	ObjectName
Object inside a Tab	TabName.ObjectName
Object inside a GroupBox (on the panel)	GroupBoxTitle.ObjectName
Object inside a GroupBox inside a Tab	TabName.GroupBoxTitle.ObjectName

In all cases, ObjectName is the name string given to the object in the cfg file — the first argument after the widget keyword. TabName is the tab label exactly as written in the Tab command. GroupBoxTitle is the title string exactly as written in the GroupBox command.

> **Important:** All names are case-sensitive and must match the cfg file exactly, including spaces. A mismatch will cause mips.Command() to return '?'.

Examples:

```
// Object 'Voltage' placed directly on the panel:
mips.Command("Voltage")

// Object 'Voltage' placed on a tab named 'DC Bias':
mips.Command("DC Bias.Voltage")

// Object 'Voltage' inside a GroupBox titled 'Channel 1' on tab 'DC Bias':
mips.Command("DC Bias.Channel 1.Voltage")
```

The general script patterns for reading, writing, and triggering any object are:

```
// Read a value or property
value = mips.Command("address").trim()

// Write a value or property
```

```
mips.Command("address=newValue")

// Trigger an action (e.g. press a button)
mips.Command("address")
```

Note: All objects return '?' when a command path is not recognised. Always use .trim() when reading string values to remove any trailing whitespace or newline characters.

14.1 DCBchannel

A DC bias channel with a setpoint (editable) and a readback (read-only).

Command / Syntax	Description
`ObjectName`	Read: returns the current setpoint voltage string.
`ObjectName=value`	Write: sets the setpoint voltage and sends the MIPS command.
`ObjectName.readback`	Read: returns the current readback (measured) voltage string.
`ObjectName.channel`	Read: returns the hardware channel number as a string.
`ObjectName.mips`	Read: returns the MIPS system name this channel is bound to.
`ObjectName.color=cssColor`	Write: sets the background color of the channel widget (e.g. yellow, pink, transparent).
`ObjectName.hide=TRUE/FALSE`	Write: shows or hides the channel widget.

14.2 DCBoffset

A DC bias offset/range control.

Command / Syntax	Description
`ObjectName`	Read: returns the current offset voltage string.
`ObjectName=value`	Write: sets the offset voltage.

14.3 DCBenable

A checkbox that enables or disables all DC bias outputs on the associated MIPS system.

Command / Syntax	Description
`ObjectName`	Read: returns ON or OFF.
`ObjectName=ON`	Write: enables DC bias power supply.
`ObjectName=OFF`	Write: disables DC bias power supply.

14.4 RFchannel

Simplified RF channel widget (drive, frequency, readbacks). Read/write Drive and Freq; all others are read-only readbacks.

Command / Syntax	Description
`ObjectName.Drive`	Read: returns the drive level (%).
`ObjectName.Drive=value`	Write: sets the drive level and sends the MIPS command.
`ObjectName.Freq`	Read: returns the drive frequency (Hz).
`ObjectName.Freq=value`	Write: sets the drive frequency.
`ObjectName.RF+`	Read: returns the RF+ peak-to-peak voltage (read-only readback).
`ObjectName.RF-`	Read: returns the RF- peak-to-peak voltage (read-only readback).
`ObjectName.Power`	Read: returns the RF power in Watts (read-only readback).
`ObjectName.color=cssColor`	Write: sets the background color of the widget.

14.5 RFCchannel

RF channel with full closed-loop control. Adds Setpoint (closed-loop voltage target) to the properties available on RFchannel.

Command / Syntax	Description
`ObjectName.Drive`	Read/Write: drive level (%). Active in open-loop mode only.
`ObjectName.Setpoint`	Read/Write: closed-loop RF voltage setpoint (Vp-p). Active in closed-loop mode only.
`ObjectName.Freq`	Read/Write: drive frequency (Hz).
`ObjectName.RF+`	Read: RF+ peak-to-peak voltage (readback, read-only).
`ObjectName.RF-`	Read: RF- peak-to-peak voltage (readback, read-only).
`ObjectName.Power`	Read: RF power in Watts (readback, read-only).
`ObjectName.color=cssColor`	Write: sets the widget background color.

14.6 DIOchannel

A digital I/O channel checkbox. Read returns 1 (checked/high) or 0 (unchecked/low).

Command / Syntax	Description
`ObjectName`	Read: returns 1 if checked, 0 if unchecked.

Command / Syntax	Description
`ObjectName=1`	Write: sets the channel high (checks the box and sends the MIPS command).
`ObjectName=0`	Write: sets the channel low.

14.7 ESIchannel

An ESI (electrospray) high-voltage channel with setpoint, readback, and enable.

Command / Syntax	Description
`ObjectName`	Read: returns the setpoint voltage string.
`ObjectName=value`	Write: sets the ESI voltage setpoint.
`ObjectName.readback`	Read: returns the measured readback voltage.
`ObjectName.ena`	Read: returns ON or OFF.
`ObjectName.ena=ON`	Write: enables the ESI channel.
`ObjectName.ena=OFF`	Write: disables the ESI channel.

14.8 ARBchannel

ARB waveform generator channel. All named fields match the labels shown on the widget.

Command / Syntax	Description
`ObjectName.Frequency`	Read/Write: waveform frequency.
`ObjectName.Amplitude`	Read/Write: waveform amplitude (Vp-p).
`ObjectName.Aux output`	Read/Write: auxiliary output level (V).
`ObjectName.Offset output`	Read/Write: DC offset output (V).
`ObjectName.Waveform`	Read/Write: waveform type (e.g. SIN, RAMP). Write finds the matching ComboBox entry.
`ObjectName.Forward`	Read/Write: TRUE if Forward radio button is selected.
`ObjectName.Reverse`	Read/Write: TRUE if Reverse radio button is selected.

14.9 Ccontrol

Ccontrol has two distinct operating modes that affect how scripts interact with it. Understanding which mode a control is in is essential before reading or writing its value from a script.

Hardware-bound controls have a MIPSname that matches a connected MIPS system. They poll hardware automatically on every update cycle. When a script writes to a hardware-bound control the value is immediately forwarded to MIPS via the configured set command, exactly as if the user had typed it. When a script reads from it, the current setpoint (not the hardware readback) is returned.

UI-only controls have a MIPSname that does not match any connected system. They hold whatever value was last written — by the user or by a script. Scripts use these freely as input fields (the user fills in a parameter, the script reads it) or as output displays (the script writes a calculated result, the user sees it). No hardware communication occurs.

> **Scripting tip:** The same mips.Command() syntax works for both modes. You do not need to know which mode a control is in to read or write it. The difference is only in what happens next: hardware-bound controls forward values to MIPS, UI-only controls store them locally.

Shared properties (all Ccontrol types)

Command / Syntax	Description
`ObjectName.script=Name`	Write: attaches the named script to fire when this control changes.
`ObjectName.scriptCall=expr`	Write: sets the call expression used when the attached script fires.
`ObjectName.color=cssColor`	Write: sets the widget background color.
`ObjectName.hide=TRUE/FALSE`	Write: shows or hides the widget.
`ObjectName.toolTip=text`	Write: sets the tooltip.
`ObjectName.getCmd`	Read: returns the current get command string. For UI-only controls this is the descriptive label set in the cfg file.
`ObjectName.getCmd=CMD`	Write: dynamically replaces the get command. Can be used to retarget a control to a different MIPS command at runtime.
`ObjectName.setCmd`	Read: returns the current set command string.
`ObjectName.setCmd=CMD`	Write: dynamically replaces the set command.
`ObjectName.rbCmd`	Read: returns the current readback command string.
`ObjectName.rbCmd=CMD`	Write: dynamically replaces the readback command.
`ObjectName.mips`	Read: returns the MIPSname string (the system name or UI-only keyword).
`ObjectName.mips=name`	Write: rebinds the control to a different MIPS system at runtime.
`ObjectName.script`	Read: returns the currently attached script name.

LineEdit

Displays a label, an editable setpoint field, and optionally a read-only readback field. The most common Ccontrol type for both hardware values and script parameters.

Command / Syntax	Description
`ObjectName`	Read: returns the current setpoint field text. For UI-only controls this is whatever value was last set by the user or a script. For hardware-bound controls it is the last value polled from MIPS.
`ObjectName=value`	Write: sets the setpoint field text. Hardware-bound: also sends the set command to MIPS immediately. UI-only: stores the value locally for the user to see or for scripts to retrieve later.
`ObjectName.readback`	Read: returns the readback field text. Only valid when a readback command is configured. For UI-only controls this field is never updated automatically — a script must write to it explicitly using ObjectName=value if a readback command is absent.

Common scripting patterns for LineEdit:

```
// Read a user-entered parameter (UI-only control)
var period = parseFloat(mips.Command("Tab.Period").trim())

// Write a script-calculated result to a display field (UI-only)
mips.Command("Tab.A+B=" + (a + b))

// Read current hardware setpoint (hardware-bound)
var freq = parseFloat(mips.Command("Tab.Rate").trim())

// Write a new value to hardware (hardware-bound, sends MIPS command)
mips.Command("Tab.Rate=1000")
```

CheckBox

Hardware-bound: polls hardware state and maps the response to checked/unchecked using the readback suffix. Checking/unchecking immediately sends the configured set or get command to MIPS. UI-only: acts as a simple boolean flag that scripts and the user can toggle freely.

Command / Syntax	Description
`ObjectName`	Read: returns TRUE if checked, FALSE if unchecked. If a readback suffix is configured (e.g. _TRUE_FALSE) returns the mapped hardware string instead.
`ObjectName=TRUE`	Write: checks the box. Hardware-bound: sends the set command to MIPS.
`ObjectName=FALSE`	Write: unchecks the box. Hardware-bound: sends the get command to MIPS.
`ObjectName=mappedValue`	Write: for controls with a readback suffix, set using the mapped value directly (e.g. =HIGH or =LOW).

Common scripting patterns for CheckBox:

```
// Read a UI-only boolean flag set by the user
var useExtTrig = mips.Command("Tab.Ext trig").trim() == "TRUE"
```

```
// Conditionally enable hardware based on checkbox state
if (mips.Command("Tab.Enable").trim() == "TRUE") {
  mips.SendCommand("MIPS-A", "SDCPWR,ON\n")
}
```

Button

Hardware-bound: pressing sends the set command to MIPS. UI-only: pressing fires the attached script (via ScriptButton) or does nothing if no script is attached. Buttons have no stored value.

Command / Syntax	Description
ObjectName	Trigger: simulates a button press. Hardware-bound: sends the set command to MIPS. Fires any attached script change event.

ComboBox

Hardware-bound: polls the hardware for the current selection and sends the set command with the selected item text when the user changes it. UI-only: acts as a mode selector whose current value scripts can read. Populate items with ComboBoxList immediately after the Ccontrol line.

Command / Syntax	Description
ObjectName	Read: returns the currently selected item text.
ObjectName=text	Write: selects the matching item. Returns '?' if the text is not found in the list. Hardware-bound: also sends the set command to MIPS with the selected text.

Common scripting pattern for ComboBox:

```
// Branch behaviour based on user mode selection
var mode = mips.Command("Tab.Type").trim()
if (mode == "Pulse") { /* ... */ }
else if (mode == "Ramp") { /* ... */ }
```

14.10 Table

An editable data table widget.

Command / Syntax	Description
ObjectName.Rows	Read: returns the number of rows as a string.
ObjectName.Columns	Read: returns the number of columns as a string.
ObjectName.Header,col	Read: returns the header text of column col (0-based).
ObjectName.Header,col=text	Write: sets the header text of column col.
ObjectName.Cell,row,col	Read: returns the cell value at (row, col) — both 0-based.

Command / Syntax	Description
`ObjectName.Cell,row,col=value`	Write: sets the cell value at (row, col).
`ObjectName.sort`	Trigger: sorts the table by column 0 in ascending order.
`ObjectName.script=Name`	Write: attaches a script to fire whenever any cell changes.
`ObjectName.scriptCall=expr`	Write: sets the call expression for the change script.

14.11 Slider

A horizontal or vertical slider widget.

Command / Syntax	Description
`ObjectName`	Read: returns the current slider value as a string.
`ObjectName=value`	Write: sets the slider position. Must be within the configured min/max range.
`ObjectName.script=Name`	Write: attaches a script to fire when the slider value changes.
`ObjectName.scriptCall=expr`	Write: sets the call expression for the change script.
`ObjectName.toolTip=text`	Write: sets the tooltip.
`ObjectName.color=cssColor`	Write: sets the slider background color.

14.12 StatusLight

A tri-color status indicator light (green, red, yellow).

Command / Syntax	Description
`ObjectName`	Read: returns the current lit color — GREEN, RED, YELLOW, or NONE.
`ObjectName=GREEN`	Write: turns on the green light (turns off others).
`ObjectName=RED`	Write: turns on the red light.
`ObjectName=YELLOW`	Write: turns on the yellow light.
`ObjectName.message`	Read: returns the current status message string.
`ObjectName.message=text`	Write: sets and displays a status message.
`ObjectName.mode`	Read: returns the current display mode.
`ObjectName.mode=value`	Write: sets the display mode.

14.13 PlotData

An embedded plot widget. All plot commands are forwarded directly to the plot engine. Use the mips.Command() set syntax:

```
mips.Command("Tab.PlotName=PlotCommand")
```

Any valid plot command string (see Section 13.4) can be sent this way. Reading the path with no argument (mips.Command("Tab.PlotName")) returns an empty string — PlotData is write-only from a script perspective.

14.14 ScriptButton

A button that runs a named script when pressed.

Command / Syntax	Description
`ObjectName`	Trigger: simulates pressing the button (runs the associated script).
`ObjectName=ABORT`	Write: aborts the script if it is currently running.
`ObjectName.scriptText`	Read: returns the script source text.
`ObjectName.scriptText=text`	Write: replaces the script source text.
`ObjectName.scriptCall`	Read: returns the current scriptCall expression.
`ObjectName.scriptCall=expr`	Write: sets the scriptCall expression.
`ObjectName.scriptAbort`	Read: returns the scriptAbort expression.
`ObjectName.scriptAbort=script`	Write: sets inline script code to execute when this button's script is aborted.

14.15 TimingGenerator

The TIMING widget exposes its full editor UI through mips.Command(), allowing scripts to configure events, set clock and trigger options, generate the sequence, and control execution entirely without user interaction. See Section 6.11 for a full description of the editor and all parameters.

Frame Parameters

Command / Syntax	Description
`TGName.Frame.Start`	Read/Write: start of the data acquisition frame in clock cycles.
`TGName.Frame.Width`	Read/Write: width of the data acquisition frame in clock cycles.
`TGName.Frame.Accumulations`	Read/Write: number of times the sequence repeats per triggered run.
`TGName.Frame.Enable`	Read/Write: DIO output to assert during the acquisition frame (A–P, or empty for none).

Clock and Trigger

Command / Syntax	Description
`TGName.Clock source`	Read/Write: Ext, ExtN, ExtS, 42000000, 10500000, 2625000, or 656250.
`TGName.Trigger source`	Read/Write: Software, Edge, Pos, or Neg.
`TGName.Mux order`	Read/Write: Hadamard mux order (None for standard operation).
`TGName.Ext Clock Freq`	Read/Write: external clock frequency in Hz (used for mS time conversion).
`TGName.Time mode, in mS`	Read/Write: TRUE enables millisecond entry mode, FALSE uses raw clock cycles.

Event Management

Command / Syntax	Description
`TGName.Event.Select event=Name`	Write: selects the named event for subsequent Event property reads and writes. Use Name=New event to add an event, Name=Delete current to remove the selected one.
`TGName.Event.Signal`	Read/Write: DIO signal channel for the selected event. Populated from available MIPS outputs.
`TGName.Event.Start`	Read/Write: start time for the selected event. Accepts a fixed number or a linked expression (e.g. Inject time.Start - Trap time.Width).
`TGName.Event.Width`	Read/Write: duration for the selected event. Accepts a fixed number or a linked expression.
`TGName.Event.Value`	Read/Write: active output value for the selected event.
`TGName.Event.Value,off`	Read/Write: inactive output value for the selected event.
`TGName.Event.StartT,Name`	Read: compiled start time in clock ticks for the named event (after Generate).
`TGName.Event.WidthT,Name`	Read: compiled width in clock ticks for the named event (after Generate).
`TGName.Event.Channel,Name`	Read: signal channel assigned to the named event.

Execution and State

Command / Syntax	Description
`TGName.Table`	Read/Write: the raw compiled pulse table string. Write to inject a table directly, bypassing the editor.
`TGName.Generate`	Trigger: compiles all events and frame parameters and downloads the table to MIPS.
`TGName.Trigger`	Trigger: sends a software trigger to start execution (only valid when Trigger source is Software).
`TGName.Abort`	Trigger: stops an active sequence run.

Command / Syntax	Description
`TGName.Edit`	Trigger: opens the timing generator editor dialog.
`TGName.isTblMode`	Read: returns TRUE if the controller is currently waiting in table mode for a hardware trigger.
`TGName.script=Name`	Write: attaches a named script to fire on timing generator state changes.

Scripting Example — Configuring and Running a Sequence

This example configures the Inject time event, sets software trigger mode, generates the sequence, and fires it:

```
// Select the Inject time event and update its start position
mips.Command("TG.Event.Select event=Inject time")
mips.Command("TG.Event.Start=200")
mips.Command("TG.Event.Width=9")

// Set clock to internal 10.5 MHz, software trigger
mips.Command("TG.Clock source=10500000")
mips.Command("TG.Trigger source=Software")
mips.Command("TG.Frame.Accumulations=10")

// Compile and download the sequence, then fire
mips.Command("TG.Generate")
mips.delay(100)
mips.Command("TG.Trigger")
```

Scripting Example — Scanning Inject Time

This example runs a loop that steps the Inject time start across a range and triggers a run at each position:

```
for (var t = 100; t <= 500; t += 50) {
  mips.Command("TG.Event.Select event=Inject time")
  mips.Command("TG.Event.Start=" + t)
  mips.Command("TG.Generate")
  mips.delay(50)
  mips.Command("TG.Trigger")
  mips.delay(500)   // wait for accumulations to complete
}
```

14.16 EventControl

A lightweight control that binds directly to a single named event value in the most recently generated timing table. Defined in the cfg with EventControl,name,eventName,x,y.

Command / Syntax	Description
`ObjectName`	Read: returns the current event value string.
`ObjectName=value`	Write: sets the event value and sends the MIPS STBLVLT command to update the live table.

14.17 TextMessage (TextLabel)

A static or dynamic text label placed on the panel.

Command / Syntax	Description
`ObjectName`	Read: returns the current label text.
`ObjectName=text`	Write: sets and displays new label text.

14.18 CPbutton

A button that reloads the control panel from a specified cfg file.

Command / Syntax	Description
`ObjectName`	Trigger: simulates pressing the button, causing the panel to reload from the configured cfg file.

14.19 DACchannel

A general-purpose DAC output channel. Reads from GDACV and writes to SDACV on the MIPS controller.

Command / Syntax	Description
`ObjectName`	Read: returns the current DAC output value string.
`ObjectName=value`	Write: sets the DAC output value.

15. Multi-Pass Compressor Control Table

Both the Twave and ARB tabs include a Multi-Pass Compressor Control Table. This text string defines a sequence of waveform states and parameter changes synchronized with a multi-pass ion compressor. All commands are case-sensitive.

15.1 State Change Commands

Command / Syntax	Description
`N`	Execute a normal (non-compressed) Twave/ARB cycle.
`C`	Execute a compression cycle.
`D`	Delay: hold the current state and stop the clock.
`[`	Start of a loop.
`]n`	End of a loop; repeat the enclosed block n times.

15.2 Parameter Commands

Command / Syntax	Description
`V`	Twave/ARB channel 1 peak-to-peak voltage (V).
`v`	Twave/ARB channel 2 peak-to-peak voltage (V).
`F`	Waveform frequency.
`O`	Compression order. O0 = infinite.
`c`	Compression cycle time in milliseconds.
`n`	Normal cycle time in milliseconds.
`S`	Output switch state: S1=close, S0=open.
`s`	Stop the Twave/ARB clock.
`r`	Restart the Twave/ARB clock.
`o`	Gate open time in milliseconds.
`g`	Time from table start to gate open (ms).
`G`	Time from table start to gate close (ms).
`M`	Compressor mode.

15.3 Examples

Two compression cycles then two normal cycles:

```
C2N2
```

Set parameters, compress, adjust, compress, two normal cycles:

```
c200v30O5Cv50CN2
```

Loop 10 times [normal, compress, compress, normal], then one normal:

```
C[NCCN]10N
```

16. Firmware Updates

> **Important:** Disconnect RF heads from the MIPS controller before any firmware operation.

16.1 Saving Current Firmware

13. Connect to MIPS and confirm communication (send GVER in Terminal).
14. Disconnect all RF heads.
15. Select Tools > Save Current MIPS Firmware and follow the prompts.

16.2 Programming New Firmware

16. Obtain the firmware .bin file from the GAA Google Drive.
17. Exit and restart the MIPS application.
18. Connect to MIPS and disconnect all RF heads.
19. Select Tools > Program MIPS, choose the .bin file, and follow the prompts.
20. The controller reboots automatically when complete.

16.3 Recovery from a Failed Update

21. Click Disconnect, then exit and restart the application.
22. The COM port may have changed (bootloader presents a different USB device). Reconfigure the port.
23. Click Connect. Timeout errors in the status bar are expected.
24. Select the Terminal tab, then use Tools > Program MIPS to upload firmware again.

17. MIPS Command Reference

Commands are sent via the Terminal tab or through scripts using mips.SendCommand() / mips.SendMess(). All commands are case-sensitive. Arguments follow the command name separated by commas. The full list below is sourced from MIPScommands.txt included with the MIPS application.

17.1 General Commands

Command / Syntax	Description
`NOP`	Do nothing
`GVER`	Report firmware version
`ISPWR`	Report power status
`SUPPLIES`	Report main supply voltages
`V12`	Report 12 volt supply
`V24`	Report 24 volt supply
`CUR`	Report input current, in amps
`PWR`	Report input power in watts
`UPTIME`	Report uptime in mins
`GERR`	Report the last error code
`GNAME`	Report MIPS system name
`SNAME`	Set MIPS system name
`SSERWD`	Set serial watch dog time in seconds, 0 or negative to disable
`GSERWD`	Return serial watch dog time in seconds
`UUID`	Reports the microcontroller's unique ID, 128 bits, hex format
`BIMAGE`	Set MIPS boot image
`ABOUT`	Report about this MIPS system
`SMREV`	Set module rev level: board (a or b), address (hex), rev
`RESET`	Reset the controller
`STATUS`	Reports the last reboot status and time in milliseconds from boot
`SAVE`	Save MIPS configuration data to default.cfg on SD card
`GCHAN`	Report number for the selected system
`MUTE`	Turns on and off the serial response from the MIPS system
`ECHO`	Turns on/off serial echo mode — command is echoed to host, TRUE or FALSE
`TRIGOUT`	Generates output trigger on rev 2+ controllers. Supports HIGH, LOW, PULSE
`AUXOUT`	Generates output trigger on Aux output. Supports HIGH, LOW, PULSE

Command / Syntax	Description
DELAY	Generates a delay in milliseconds (used by macro functions)
GCMDS	Send a list of all commands
GAENA	Print the UseAnalog flag, true or false
SAENA	Sets the UseAnalog flag, true or false
THREADS	List all threads, their IDs, and last runtimes
THREAD	List details on the named thread
STHRDENA	Set thread enable to true or false
STHRDINT	Set thread run interval in mS: name, interval (1 to 10000)
RUNNOW	Run the thread defined by name, now
THRDRESTART	Restart all threads
TBLTSKENA	Enables tasks in table mode, true or false
LEDOVRD	Override the LED operation if true, always false on startup
LED	Define the LED state you are looking for
DSPOFF	Display off command
STRTDLY	Startup delay in seconds
SSERIALNAV	Set flag to TRUE to enable UI navigation from the host interface
SSPND	Suspend all tasks (suspends real time control), TRUE or FALSE
GSPND	Returns suspend status, TRUE or FALSE
CPUTEMP	Returns the CPU temp in degrees C
USBPWR	Enables or disables USB powering of controller. TRUE = enable
DREAD	Read an Arduino pin, returns HIGH or LOW
DWRITE	Write an Arduino pin: HIGH, LOW, PULSE, or SPLUSE
DSET	Sets an Arduino pin mode: INPUT, PULLUP, OUTPUT
STRPLVL	Set the DCbias power supply readback error trip point in percentage of FS
GTRPLVL	Returns the DCbias power supply readback error trip point
POWER	Cycle MIPS system power
STBLRETRIG	Set the table retriggerable flag, TRUE = retriggerable
GTBLRETRIG	Returns the table retriggerable flag

17.2 SD Card and File Commands

Command / Syntax	Description
DIR	List all files on the SD card
CREATE	Create an empty file, fails if file is present

Command / Syntax	Description
ADD	Add a string to the filename passed
MORE	List file to serial port
READSD	Read line number from file, 0 is first line
DEL	Delete file on the SD card
GET	Dump file contents, hex, from SD card file
PUT	Create file on SD card from host interface
SAVEMOD	Save module EEPROM to SD file. Filename, Board (A or B), Address (hex)
LOADMOD	Load module EEPROM from SD file. Filename, Board (A or B), Address (hex)
SAVEALL	Saves all modules EEPROM data to the SD card
LOADALL	Loads all modules EEPROM data from the SD card
GETEEPROM	Sends selected EEPROM data to the host. Board (A or B), Address (hex)
PUTEEPROM	Receives data from the host and writes to EEPROM. Board (A or B), Address (hex)
CHKIMAGE	Reports the image file status
LOADIMAGE	Loads an image file to the display
MRECORD	Turn on macro recording into the filename argument
MSTOP	Stop macro recording and close the file
MPLAY	Play a macro file
MLIST	Send a list of macro files
MDELETE	Delete a macro file

17.3 TWI (I2C) Commands

Command / Syntax	Description
TWIRESET	Resets the TWI interface
TWIERROR	Reports the number of detected TWI failures
STWIHDW	If true, the TWI hardware interface is used for ADC read functions
GTWIHDW	Returns the current TWI hardware status
TWICMD	Sends a command to the addressed TWI (Wire) device. Args: board, address (decimal), string
TWI1CMD	Sends a command to the addressed TWI1 (Wire1) device. Args: address (decimal), string
TWISCAN	Scans the selected board address for TWI devices (Wire interface)
TWI1SCAN	Scans the selected board address for TWI devices (Wire1 interface)

Command / Syntax	Description
TWIADDSET	Automatically sets the TWI addresses for the selected module: RFD, DCB, FAIMS
TWITALK	Redirect serial communications through a board and TWI address
TWI1TALK	Redirect serial communications through a board and TWI1 address
STWISPEED	Set TWI speed: 0 or 1
GTWISPEED	Returns TWI speed: 0 or 1
GTWI	Reads value from TWI device
STWI	Writes a value to a TWI device

17.4 Log and Clock Commands

Command / Syntax	Description
SLOGENA	Enables logging if TRUE, disables if FALSE
GLOGENA	Returns the log enable status
LOGREP	Reports the current log entries
STIME	Sets the current time, 00:00:00 format
GTIME	Returns the current time, 00:00:00 format
SDATE	Sets the current date, dd/mm/yyyy format
GDATE	Returns the current date, dd/mm/yyyy format

17.5 Pulse Counter and Clock Generation

Command / Syntax	Description
COUNT	Setup pulse counter. Args: channel (Q-X or NA), level (POS, NEG, BOTH, NA)
GCNT	Returns the current pulse count
CLRCNT	Clear the current count
SCNTTRG	Set the counter threshold count
GCNTTRG	Return the counter threshold count
TRIGCNT	If TRUE generate 5 uS trigger on Trig out at threshold count
TRIGRST	If TRUE reset counter at threshold count
GWIDTH	Report the pulse width in microseconds
SWIDTH	Set the pulse width in microseconds
GFREQ	Report the pulse frequency in Hz
SFREQ	Set the pulse frequency in Hz
BURST	Generates a frequency burst on trig out line

Command / Syntax	Description
GMAXFTRIG	Report the maximum frequency for trig out, 0 = no limit
SMAXFTRIG	Sets the maximum frequency for trig out, 0 = no limit
GMAXFATRIG	Report the maximum frequency for aux trig out, 0 = no limit
SMAXFATRIG	Sets the maximum frequency for aux trig out, 0 = no limit
SDTRIGINP	Set delay trigger input (Q-X) and level, POS or NEG
SDTRIGDLY	Set trigger delay in uS
GDTRIGDLY	Returns trigger delay in uS
SDTRIGPRD	Set trigger delay repeat period in uS
GDTRIGPRD	Returns trigger delay repeat period in uS
SDTRIGRPT	Defines the number of trigger repeats, 0 = forever
GDTRIGRPT	Returns the number of trigger repeats
SDTRIGMOD	Defines the delay trigger module: ARB, ADC, SWEEP, AUXTRIG
SDTRIGENA	TRUE enables the trigger, FALSE disables
GDTRIGENA	Returns the trigger delay enable status

17.6 ADC Commands

Command / Syntax	Description
ADC	Read and report ADC channel. Valid range 0 through 3
ADCAVG	Report the average ADC value read by the change detection function
ADCGAIN	Set ADC channel gain. Channel range 0 to 3, gains are 1, 2, or 4
ADCCHG	Enable ADC change detection. Channel range 0 to 3, threshold 0 to 100
ADCABORT	ADC system abort
SADCCHAN	Set ADC channel number
GADCCHAN	Get ADC channel number
SADCSAMPS	Set ADC number of samples
GADCSAMPS	Get ADC number of samples
SADCVECTS	Set ADC number of vectors
GADCVECTS	Get ADC number of vectors
SADCRATE	Set ADC sample rate in Hz
GADCRATE	Get ADC sample rate in Hz
ADCRBINIT	ADC record vector initialization
ADCRBTRIG	ADC record vector trigger
ADCRBREAD	ADC record vector read. Args: start, num

Command / Syntax	Description
ADCRBSUM	Reports the sum of all values in the recorded vector
ADCRBMAX	Reports the maximum value in the recorded vector
ADCRBVECS	Reports the number of vectors recorded

17.7 DC Bias Commands

Command / Syntax	Description
SDCB	Set voltage value
GDCB	Get voltage setpoint value
GDCBV	Get actual measured voltage value
SDCBOF	Set float voltage for selected board
GDCBOF	Read float voltage for selected board
GDCMIN	Read minimum voltage for selected board
GDCMAX	Read maximum voltage for selected board
SDCPWR	Sets the DC bias power: ON or OFF
GDCPWR	Get the DC bias power status: ON or OFF
SDCBALL	Sets the DCbias setpoints for all outputs
GDCBALL	Returns the DC bias voltage setpoints for all channels in the system
GDCBALLV	Returns the DC bias voltage readback values for all channels
SDCBDELTA	Set all DC bias channels by a delta value
SDCBUPDATE	Set to TRUE to force all DC bias channels to update
GDCBCAL	Returns the selected channel's calibration parameters
SDCCALM	Set the DC bias channel cal parameter M
SDCCALB	Set the DC bias channel cal parameter B
SDCBPRO	Sets a DC bias profile
GDCBPRO	Reports the selected DC bias profile
ADCBPRO	Applies the selected DC bias profile
CDCBPRO	Copy the current DC bias values to the selected profile
TDCBPRO	Enables toggling between two profiles with user defined dwell time in mS
TDCBSTP	Stop the profile toggling
SDCBOFOF	Set the DC bias global offset, applies to all channels on module
GDCBOFOF	Returns the DC bias global offset
SDCBCHOF	Set the DC bias channel offset, applies to all enabled channels on module
GDCBCHOF	Returns the DC bias channel offset

Command / Syntax	Description
SDCBCHMK	Set the DC bias channel offset mask (hex), defines channels to apply offset
GDCBCHMK	Returns the DC bias channel offset mask (hex)
SDCBPCH	Set DC bias pulse channel
GDCBPCH	Get DC bias pulse channel
SDCBPV	Set DC bias pulse voltage in volts
GDCBPV	Get DC bias pulse voltage in volts
SDCBPD	Set DC bias pulse delay in uS
GDCBPD	Get DC bias pulse delay in uS
SDCBPW	Set DC bias pulse width in uS
GDCBPW	Get DC bias pulse width in uS
SDCBPT	Set DC bias pulse trigger source: Q-X or t
GDCBPT	Get DC bias pulse trigger source
SDCBPENA	Set DC bias pulse enable: TRUE or FALSE
GDCBPENA	Get DC bias pulse enable status
WFMINIT	Initialize waveform generation and define samples per second
WFMADDWF	Define a waveform. Args: channel, freq, min, max
WFMENA	Enable waveform generation
WFMDIS	Disable waveform generation
DSTATE	Define a state. Args: name, channel, value...
SSTATE	Sets DCbias channels to the values defined in the named state
LSTATES	List all defined state names
RSTATE	Remove a state from the linked list
RSTATES	Clear the full linked list of states
GSTATE	Returns true if named state is in list, else false
DSEGMENT	Defines a segment. Args: name, length, next, repeat count
ADDSEGTP	Adds a time point to a segment. Args: name, count, state1, state2...
ADDSEGTRG	Adds a trigger point to a segment. Args: name, count, port, level
ADDSEGSTRG	Defines the start trigger for a segment. Args: name, source, level
PSEGMENTS	Execute the segment list
SABORT	Abort executing the segment list
TSEGMENT	Software trigger the segment if ready
CDCBCH	Calibrate the requested DC bias channel
CDCBCHS	Calibrate all DC bias channels in order
CDCBOFF	Calibrate all DC bias offsets for channel number

17.8 RF Driver Commands

Command / Syntax	Description
SRFFRQ	Set RF frequency
SRFVLT	Set RF output voltage
SRFDRV	Set RF drive level
GRFFRQ	Report RF frequency
GRFPPVP	Report RF output voltage, positive phase
GRFPPVN	Report RF output voltage, negative phase
GRFDRV	Report RF drive level in percentage
GRFVLT	Report RF output voltage setpoint
GRFPWR	Report RF head power draw
GRFMODE	Report RF mode: MANUAL or AUTO for selected channel
SRFMODE	Sets the RF mode: MANUAL or AUTO for selected channel
GRFALL	Reports Freq, RFVpp+ and - for each RF channel in system
TUNERFCH	Auto tune the selected RF channel
RETUNERFCH	Auto retune the selected RF channel
SRFCAL	Sets the RF calibration parameters. Args: channel, slope, intercept
SRFPL	Sets the RF power limit for the given channel in watts
GRFPL	Returns the RF power limit for the given channel in watts
TUNEABORT	Auto tune abort
SRFATMINF	Set auto tune start frequency
GRFATMINF	Return auto tune start frequency
SRFATMAXF	Set auto tune stop frequency
GRFATMAXF	Return auto tune stop frequency

17.9 RF Amplifier / QUAD Commands

Command / Syntax	Description
SRFAENA	Sets the RF system enable mode: ON or OFF
GRFAENA	Returns the RF system enable mode: ON or OFF
SRFAFREQ	Sets the RF system frequency
GRFAFREQ	Returns the RF frequency
SRFAK	Sets the resolving DC voltage ratio
GRFAK	Returns the resolving DC voltage ratio
SRFAMOD	Sets the RF system to open or closed loop
GRFAMOD	Returns the RF mode state: open or closed

Command / Syntax	Description
SRFADRV	Sets the RF drive level, 0 to 100%
GRFADRV	Returns the RF drive level
SRFALEV	Sets the RF output voltage setpoint
GRFALEV	Returns the RF output voltage setpoint
GRFAVPPA	Returns the RF output A actual level
GFRAVPPB	Returns the RF output B actual level
GRFAPWR	Returns the RF amp RF forward power
SRFARNG	Sets the QUAD RF maximum RF level
GRFARNG	Returns the QUAD RF maximum RF level
SRFAPB	Sets the pole bias DC
GRFAPB	Returns the pole bias DC
SRFAR0	Sets the Ro value in mm
GRFAR0	Returns the Ro value in mm
SRFAMZ	Sets the m/z in amu
GRFAMZ	Returns the m/z in amu
SRFARES	Sets the resolution in AMU
GRFARES	Returns the resolution in AMU
RFAQUPDATE	Updates the QUAD parameters
RFAACQ	Report last point and acquire. Args: module, mz, delta, dwell
SRFAGAIN	Sets RF head level control gain: HIGH or LOW
GRFAGAIN	Returns RF head level control gain: HIGH or LOW
SRFATUNE	Start auto tune for selected module

17.10 DIO Module Commands

Command / Syntax	Description
SDIO	Set DIO output bit
GDIO	Get DIO output bit
RPT	Report an input state change
SHOLDOFF	Set DIO change reporting holdoff in mS
GHOLDOFF	Report DIO change reporting holdoff in mS
MIRROR	Mirror an input to an output
MDIO	Monitors a digital input for a state change
RDIO	Returns true if a state change was detected, else false
SDIINV	Sets digital input inversion mask, radix 10

Command / Syntax	Description
GDIINV	Returns digital input inversion mask, radix 10

17.11 ESI (High Voltage) Commands

Command / Syntax	Description
SHV	Set channel high voltage
GHV	Returns the high voltage setpoint
GHVV	Returns the actual high voltage output
GHVI	Returns the output current in mA
GHVMAX	Returns the maximum high voltage output value
GHVMIN	Returns the minimum high voltage output value
SHVENA	Enables a selected channel
SHVDIS	Disables a selected channel
GHVSTATUS	Returns the selected channel's status: ON or OFF
SHVRAMP	Set ESI voltage ramp rate. Units are V/0.1s
GHVRAMP	Returns ESI voltage ramp rate for the selected module
SHVGATE	Sets the gating enable flag for selected channel
GHVGATE	Returns the gating enable flag for selected channel

17.12 Pulse Table Commands

Command / Syntax	Description
STBLDAT	Set the pulse sequence table data
STBLCLK	Clock mode: EXT or INT
STBLTRG	Trigger mode: EXT or SW
TBLABRT	Abort table operation
SMOD	Set the table mode (e.g. SMOD,TBL or SMOD,ONCE)
TBLSTRT	Set the software trigger
TBLSTOP	Stops a running table
GTBLFRQ	Returns the current internal clock frequency
STBLNUM	Set the active table number
GTBLNUM	Get the active table number
STBLADV	Set the table advance status: ON or OFF
GTBLADV	Get the table advance status: ON or OFF
STBLVLT	Set a value in a loaded table

Command / Syntax	Description
GTBLVLT	Get a value from a loaded table
STBLCNT	Set a count value in a loaded table
STBLDLY	Defines the inter-table delay in milliseconds
GTBLREPLY	Returns TRUE/FALSE state of enable table response flag
STBLREPLY	TRUE or FALSE — set to TRUE to enable table response messages
TBLRPT	Report table as hex bytes
STBLTSKS	Set to TRUE to enable tasks in table mode based on available time
GTBLTSKS	Returns the TblTasks flag status
SEXTFREQ	Sets the external frequency used for the table clock in Hz
GEXTFREQ	Returns the ExtFreq value in Hz
STBLVDLT	If true, voltage delta mode is enabled in table
TBLCHK	Tests a table for timing violations and prints the results
GTBLSTA	Returns the table status

17.13 TWAVE Commands

Command / Syntax	Description
GTWF	Report the TWAVE frequency
STWF	Set the TWAVE frequency
GTWPV	Report the TWAVE pulse voltage
STWPV	Set the TWAVE pulse voltage
GTWG1V	Report the TWAVE Guard 1 voltage
STWG1V	Set the TWAVE Guard 1 voltage
GTWG2V	Report the TWAVE Guard 2 voltage
STWG2V	Set the TWAVE Guard 2 voltage
GTWSEQ	Report the TWAVE sequence
STWSEQ	Set the TWAVE sequence
GTWDIR	Report the TWAVE waveform direction: FWD or REV
STWDIR	Set the TWAVE waveform direction: FWD or REV
STWCTBL	Twave compressor table definition setting command
GTWCTBL	Twave compressor table definition reporting command
GTWCMODE	Report Twave compressor mode
STWCMODE	Set Twave compressor mode
GTWCORDER	Report Twave compressor order
STWCORDER	Set Twave compressor order

Command / Syntax	Description
GTWCTD	Report Twave compressor trigger delay in mS
STWCTD	Set Twave compressor trigger delay in mS
GTWCTC	Report Twave compressor compress time in mS
STWCTC	Set Twave compressor compress time in mS
GTWCTN	Report Twave compressor normal time in mS
STWCTN	Set Twave compressor normal time in mS
TWCTRG	Force a Twave compressor trigger
GTWCSW	Report Twave compressor Switch state
STWCSW	Set Twave compressor Switch state
STWSSTRT	Set the TWAVE sweep start frequency
GTWSSTRT	Return the TWAVE sweep start frequency
STWSSTP	Set the TWAVE sweep stop frequency
GTWSSTP	Return the TWAVE sweep stop frequency
STWSSTRTV	Set the TWAVE sweep start voltage
GTWSSTRTV	Return the TWAVE sweep start voltage
STWSSTPV	Set the TWAVE sweep stop voltage
GTWSSTPV	Return the TWAVE sweep stop voltage
STWSTM	Set the TWAVE sweep time in seconds. Args: module, time
GTWSTM	Return the TWAVE sweep time
STWSGO	Start the sweep
STWSHLT	Stop the sweep
GTWSTA	Return the TWAVE sweep status

17.14 ARB Commands

Command / Syntax	Description
SARBMODE	Sets the ARB mode
GARBMODE	Reports the ARB mode
SWFREQ	Sets waveform frequency, 0 to 40000 Hz
GWFREQ	Returns the waveform frequency
SWFVRNG	Sets waveform voltage range
SWFVRAMP	Sets waveform voltage channel ramp rate in V/s
SWFVOFF	Sets waveform offset voltage
SWFVAUX	Sets waveform aux voltage
SWFDIS	Stops waveform generation

Command / Syntax	Description
SWFENA	Starts waveform generation
SWFDIR	Sets the waveform direction: FWD or REV
GWFDIR	Returns the waveform direction: FWD or REV
SWFTYP	Sets the arbitrary waveform type
GWFTYP	Returns the arbitrary waveform type
SWFARB	Sets an arbitrary waveform
GWFARB	Returns an arbitrary waveform
ARBSYNC	Issues a software sync
SARBBUF	Sets ARB buffer length
GARBBUF	Reports ARB buffer length
SARBNUM	Sets number of ARB buffer repeats per trigger
GARBNUM	Reports number of ARB buffer repeats per trigger
SARBCHS	Sets all ARB channels in the full buffer to a defined value
SARBCH	Sets a defined ARB channel in the full buffer to a defined value
SARBCTBL	ARB compressor table definition setting command
GARBCTBL	ARB compressor table definition reporting command
GARBCMODE	Report ARB compressor mode
SARBCMODE	Set ARB compressor mode
GARBCORDER	Report ARB compressor order
SARBCORDER	Set ARB compressor order
GARBCTC	Report ARB compressor compress time in mS
SARBCTC	Set ARB compressor compress time in mS
GARBCTN	Report ARB compressor normal time in mS
SARBCTN	Set ARB compressor normal time in mS
TARBTRG	Force a ARB compressor trigger
GARBCSW	Report ARB compressor Switch state
SARBCSW	Set ARB compressor Switch state
SALTTRG	Defines the alternate waveform external trigger input: Q through W or NA
SALTENA	Enables alternate waveform for selected module: TRUE or FALSE
SALTWFM	Sets the alternate waveform type: COMP, REV, ARB, FIX, CUR
GALTWFM	Returns the alternate waveform type
SARBSGO	Start the ARB sweep
SARBSHLT	Stop the ARB sweep
GARBSTA	Return the ARB sweep status
GARBVER	Returns the ARB module version number

Command / Syntax	Description
GARBPPP	Returns the ARB module points per period, 8 to 32
SARBPPP	Sets the ARB module points per period, 8 to 32
SARBEXT	Sets the selected ARB external clock source: MIPS or EXT

17.15 FAIMS Commands

Command / Syntax	Description
SFMENA	Set the FAIMS enable flag, TRUE enables waveform generation
GFMENA	Returns the FAIMS enable flag
SFMDRV	Sets FAIMS drive level in percent
GFMDRV	Returns FAIMS drive level in percent
GFMPWR	Returns FAIMS total power in watts
GFMPV	Returns FAIMS positive peak output voltage
GFMNV	Returns FAIMS negative peak output voltage
SFMLOCK	Sets FAIMS output level lock mode
GFMLOCK	Returns FAIMS output level lock mode
SFMSP	Sets FAIMS output level lock setpoint
GFMSP	Returns FAIMS output level lock setpoint
SFMFREQ	Set FAIMS frequency
GFMFREQ	Return FAIMS frequency
SFMCV	Sets FAIMS DC CV voltage setpoint
GFMCV	Returns FAIMS DC CV voltage setpoint
GFMCVA	Returns FAIMS DC CV voltage actual
SFMBIAS	Sets FAIMS DC Bias voltage setpoint
GFMBIAS	Returns FAIMS DC Bias voltage setpoint
GFMBIASA	Returns FAIMS DC Bias voltage actual
SFMOFF	Sets FAIMS DC offset voltage setpoint
GFMOFF	Returns FAIMS DC offset voltage setpoint
GFMOFFA	Returns FAIMS DC offset voltage actual
SFMCVSTART	Sets FAIMS DC CV scan start voltage
GFMCVSTART	Returns FAIMS DC CV scan start voltage
SFMCVEND	Sets FAIMS DC CV scan end voltage
GFMCVEND	Returns FAIMS DC CV scan end voltage
SFMDUR	Sets FAIMS DC CV scan duration in seconds
GFMDUR	Returns FAIMS DC CV scan duration in seconds

Command / Syntax	Description
SFMLOOPS	Sets FAIMS DC CV scan loops
GFMLOOPS	Returns FAIMS DC CV scan loops
SFMSTRTLIN	Sets FAIMS DC CV linear scan flag
GFMSTRTLIN	Returns FAIMS DC CV linear scan flag
SFMSTPTM	Sets FAIMS DC CV scan step duration
GFMSTPTM	Returns FAIMS DC CV scan step duration
SFMSTEPS	Sets FAIMS DC CV step scan number of steps
GFMSTEPS	Returns FAIMS DC CV step scan number of steps
SFMTUNE	Set auto tune request flag
SFMTABRT	Set auto tune abort flag
GFMTSTAT	Returns the auto tune state string
SFMTDRV	Set the tune mode drive level
GFMTDRV	Return the tune mode drive level
SFMATSTRF	Set the auto tune start frequency
GFMATSTRF	Return the tune mode start frequency
SFMATSTPF	Set the auto tune stop frequency
GFMATSTPF	Return the tune mode stop frequency

17.16 Filament Commands

Command / Syntax	Description
GFLENA	Get filament ON/OFF status
SFLENA	Set filament ON/OFF status
GFLI	Get filament channel current setpoint
GFLAI	Get filament channel actual current
SFLI	Set filament channel current
GFLSV	Get filament supply voltage setpoint
GFLASV	Get the actual supply side voltage
SFLSV	Set filament supply voltage
GFLV	Get filament voltage (actual)
GFLPWR	Get filament power (actual)
GFLRT	Get filament current ramp rate in amps per second
SFLRT	Set filament current ramp rate in amps per second
GFLDIR	Get filament current direction
SFLDIR	Set filament current direction

Command / Syntax	Description
GFLENAR	Get filament cycle status: OFF or number of cycles remaining
SFLENAR	Set filament cycle status: ON or OFF
GFLECUR	Returns the filament emission current

17.17 WiFi and Ethernet Commands

Command / Syntax	Description
GHOST	Report this MIPS box host name
GSSID	Report the WiFi SSID to connect to
SHOST	Set this MIPS box host name
SSSID	Set the WiFi SSID to connect to
SPSWD	Set the WiFi network password
SWIFIENA	Set the WiFi enable flag
SWIFISP	Set the WiFi serial port
ENTEST	Tests the Ethernet adapter connection
GEIP	Report the Ethernet adapter IP address
SEIP	Set the Ethernet adapter IP address
GESNIP	Report the Ethernet adapter Subnet IP address
SESNIP	Set the Ethernet adapter Subnet IP address
GEPORT	Report the Ethernet adapter port number
SEPORT	Set the Ethernet adapter port number
GEGATE	Report the Ethernet adapter gateway IP address
SEGATE	Set the Ethernet adapter gateway IP address

17.18 DAC Module Commands

Command / Syntax	Description
SDACV	Sets the named DAC channel's value
GDACV	Returns the named DAC channel's value
SDACMAX	Sets the named DAC channel's maximum
GDACMAX	Returns the named DAC channel's maximum
SDACMIN	Sets the named DAC channel's minimum
GDACMIN	Returns the named DAC channel's minimum
SDACUN	Sets the named DAC channel's units
GDACUN	Returns the named DAC channel's units

Command / Syntax	Description
SDACNM	Sets the DAC channel name defined by module and channel number
GDACMN	Returns the DAC channel name defined by module and channel number

17.19 Pulse Generator Commands

Command / Syntax	Description
SPGENA	Set Pulse Generator enable. Args: Ch, TRUE|FALSE
GPGENA	Return Pulse Generator enable. Args: Ch
SPGRET	Set Pulse Generator re-trigger. Args: Ch, TRUE|FALSE
GPGRET	Return Pulse Generator re-trigger
SPGARMT	Set Pulse Generator arm input. Args: Ch, 0|Q|R|S|T
GPGARMT	Return Pulse Generator arm input
SPGARML	Set Pulse Generator arm level. Args: Ch, CHANGE|RISING|FALLING
GPGARML	Return Pulse Generator arm level
SPGTRG	Set Pulse Generator trigger input. Args: Ch, 0|Q|R|S|T
GPGTRG	Return Pulse Generator trigger input
SPGTRGL	Set Pulse Generator trigger level. Args: Ch, CHANGE|RISING|FALLING
GPGTRGL	Return Pulse Generator trigger level
SPGSKIP	Set Pulse Generator trigger skip count. Args: Ch, Count
GPGSKIP	Return Pulse Generator trigger skip count
SPGDLY	Set Pulse Generator trigger delay in uS. Args: Ch, Delay
GPGDLY	Return Pulse Generator trigger delay in uS
SPGWDTH	Set Pulse Generator pulse width in uS. Args: Ch, Width
GPGWDTH	Return Pulse Generator pulse width in uS
SPGTRGO	Set Pulse Generator trigger output. Args: Ch, 0|A|B|C|D
GPGTRGO	Return Pulse Generator trigger output
SPGTRGF	Set Pulse Generator FET switch output. Args: Ch, 0|1|2
GPGTRGF	Return Pulse Generator FET switch output
SPGOUTCH	Set Pulse Generator DC bias output. Args: Ch, 0|1|2|3|4
GPGOUTCH	Return Pulse Generator DC bias output
SPGPV	Set Pulse Generator pulse voltage. Args: Ch, Voltage
GPGPV	Return Pulse Generator pulse voltage
SPGNUM	Set Pulse Generator number of pulses. Args: Ch, Count
GPGNUM	Return Pulse Generator number of pulses

Command / Syntax	Description
`SPGPER`	Set Pulse Generator period in uS. Args: Ch, Value
`GPGPER`	Return Pulse Generator period in uS

18. Troubleshooting

Item	Detail
Cannot connect via USB on Windows	Install the Arduino Due USB driver included with the MIPS package. Verify the COM port in Device Manager.
MSVCR110.dll missing on Windows	Run vcredist_x86.exe included in the MIPS installation package.
Cannot connect via TCP/IP	Verify the IP address or hostname and that no firewall is blocking the connection.
Module tabs do not appear after connecting	Try disconnecting and reconnecting. Check the Terminal for error messages.
MIPS unresponsive after failed firmware update	See Section 15.3. The controller is likely in bootloader mode and the COM port number will have changed.
Control panel controls show no values	Ensure MIPS systems are connected before loading the cfg file. MIPS system names in the cfg (or Define values) must match names stored in the controllers.
StartUp script errors	Check the scripting console for error messages. Common causes: wrong Tab.Control path, MIPS system name mismatch, or referencing a control before it is created.
Script locks up the application	Any loop must contain a mips.msDelay() call. Restart the application if locked.
TCPserver not accepting connections	Verify the port number is not blocked by a firewall. Only one client can connect at a time.
mips.Command() returns '?'	The addressed control path does not exist. Check Tab name, GroupBox title, and Control name for typos or case mismatches.
Async messages not received	Ensure mips.Command("MIPSname.enableAsync=TRUE") is called in StartUp and that the firmware supports async messaging.
Define name not substituting correctly	Define must appear before any Ccontrol or widget command that uses the name. Check spelling and case — substitution is case-sensitive.

Appendix A: Worked Example — Multi-System Control Panel

This appendix walks through a complete, real-world control panel cfg file. The panel controls two MIPS systems (MIPS-A and MIPS-B) and demonstrates most key cfg concepts: panel-level buttons, hardware widgets, GroupBoxes, a Tab widget, Ccontrols, Tables, ScriptButtons, and scripts including StartUp and Common.

A.1 Panel Overview

The panel drives the following hardware across two connected MIPS systems:

- 4 RF channels (2 from MIPS-A, 2 from MIPS-B)
- 2 ARB channels and a Compressor (MIPS-B)
- 16 DC bias channels split across two GroupBoxes (8 per system)
- 8 digital I/O channels split across two GroupBoxes (4 per system)
- A Tab widget with three tabs: Pulse generator A, Pulse generator B, Timing generators

A.2 Panel-Level Commands

The first block establishes the panel size, help file, standard control buttons, and hardware initialization:

```
size,960,800
Help,/Users/.../ExampleHelp.txt

Shutdown,System shutdown,20,50
SaveLoad,Save method,Load method,20,90
MIPScomms,20,170
Script,20,210

SendCommand,MIPS-A,STBLTSKS,TRUE
SendCommand,MIPS-B,STBLTSKS,TRUE
```

size sets the panel canvas dimensions in pixels. Shutdown, SaveLoad, MIPScomms, and Script add the standard utility buttons at fixed positions. The two SendCommand lines run immediately at panel load time, enabling pulse table task scheduling on both MIPS systems before any user interaction.

A.3 Hardware Channel Widgets

RF, ARB, and Compressor widgets are placed directly on the panel background with no container:

```
RFchannel,RF 1,MIPS-A,1,25,600
RFchannel,RF 2,MIPS-A,2,260,600
RFchannel,RF 3,MIPS-B,1,500,600
RFchannel,RF 4,MIPS-B,2,730,600

ARBchannel,ARB channel 1,MIPS-B,1,700,50
ARBchannel,ARB channel 2,MIPS-B,2,700,300
COMPRESSOR,Compressor,MIPS-B,750,250
```

Each widget is named (e.g. RF 1), bound to a MIPS system and channel number, and positioned at (x,y) on the panel. Because these are placed directly on the panel with no container, their script addresses are just their names. For example:

```
mips.Command('RF 1.Drive')
mips.Command('ARB channel 1.Frequency')
```

A.4 DC Bias GroupBoxes

DC bias channels are grouped inside labeled GroupBoxes, one per MIPS system. Controls inside a GroupBox use coordinates relative to the GroupBox top-left corner:

```
GroupBox,MIPS-A DCbias,250,270,190,50
  DCBchannel,CH 1,MIPS-A,1,10,20
  DCBchannel,CH 2,MIPS-A,2,10,45
  ...
  DCBoffset,Offset,MIPS-A,1,10,220
  DCBenable,DC bias enable,MIPS-A,10,245
GroupBoxEnd

GroupBox,MIPS-B DCbias,250,270,450,50
  DCBchannel,CH 1,MIPS-B,1,10,20
  ...
GroupBoxEnd
```

Because these controls are inside a GroupBox but not inside any Tab, their script addresses include the GroupBox title as a prefix:

```
// Read CH 1 setpoint from the MIPS-A DCbias group
mips.Command('MIPS-A DCbias.CH 1')

// Enable DC bias on MIPS-B
mips.Command('MIPS-B DCbias.DC bias enable=ON')
```

A.5 Tab Widget and Tab Contents

A Tab widget creates three switchable pages. Each page is populated inside a TabSelect/TabSelectEnd block. The Tab is named Tab and its label appears as the panel prefix when addressing controls:

```
Tab,Tab,500,260,190,330,Pulse generator A,Pulse generator B,Timing
generators

TabSelect,Tab,Pulse generator A
  Ccontrol,Period,Pulse,LineEdit,,period in mS,,mS,20,25
  Ccontrol,Cycles,Number,LineEdit,,of cycles,,,20,50
  Ccontrol,Ext trig,Enable,CheckBox,,external trigger on R input,,,20,75
  TextLabel,Define each pulse in table,12,20,105
  ImageBox,pulse.png,100,100,10,130
  Table,Table,10,5,350,100,130,130
  ScriptButton,Generate,generateA,250,30
  ScriptButton,Abort,abortA,250,55
TabSelectEnd
```

Objects inside a tab are addressed as TabName.ObjectName, where TabName is the tab label string. Period, Cycles, and Ext trig are UI-only controls: their MIPSnames (Pulse, Number, Enable) do not match any connected MIPS system, so they have no hardware polling. They act purely as input fields — the user fills them in and the generateA script reads them back. This is the standard pattern for capturing operator parameters in a control panel.

```
// Read the Period value from the Pulse generator A tab
mips.Command('Pulse generator A.Period')

// Read cell (0,0) from the Table on the Pulse generator B tab
mips.Command('Pulse generator B.Table.Cell,0,0')

// Trigger the Generate button on Pulse generator A
mips.Command('Pulse generator A.Generate')
```

A.6 StartUp Script

The StartUp script runs automatically once when the panel loads. It sets column headers on both Table widgets and initializes default values for Period and Cycles:

```
ScriptObj,StartUp,StartUp
Script,StartUp
  name = "Pulse generator A.Table"
  mips.Command(name + ".Header,0=Channel")
  mips.Command(name + ".Header,1=T1")
  mips.Command(name + ".Header,2=T2")
  mips.Command(name + ".Header,3=V1")
  mips.Command(name + ".Header,4=V2")
```

```
  name = "Pulse generator A"
  mips.Command(name + ".Period=250")
  mips.Command(name + ".Cycles=10")
  // ... same block repeated for Pulse generator B
EndScript
```

A JavaScript variable (name) builds the address string, avoiding repetition and making the code easier to maintain. This pattern of storing the object path prefix in a variable is a common and recommended practice.

A.7 Common Utility Functions

The Common script defines helper functions that are automatically appended to every other script on the panel:

```
Script,Common
  function getInt(a)    { return parseInt(mips.Command(a).trim()) }
  function getFloat(a)  { return parseFloat(mips.Command(a).trim()) }
  function getBool(a)   { return mips.Command(a).trim() == "TRUE" }
  function getString(a) { return mips.Command(a).trim() }
  function readTableCell(name, r, c) {
    return mips.Command(name + ".Cell," + r + "," + c).trim()
  }
  function processTable(name) { /* ... see below */ }
EndScript
```

getInt, getFloat, getBool, and getString are thin wrappers around mips.Command() that handle .trim() and type conversion automatically. processTable reads every row from a Table widget, converts time values in milliseconds to 10.5 MHz clock ticks, sorts all events by time, and assembles the STBLDAT pulse table string ready to send to MIPS.

A.8 Generate Script

The generateA script fires when the Generate ScriptButton on Pulse generator A is pressed. It reads the Table and the three Ccontrols, builds the pulse table string, and downloads it to MIPS-A:

```
Script,generateA
  mips.SendCommand("MIPS-A", "TBLABRT\n")
  res     = processTable("Pulse generator A.Table")
  period  = getFloat("Pulse generator A.Period") * 10500.0
  cycles  = getInt("Pulse generator A.Cycles")
  extTrig = getBool("Pulse generator A.Ext trig")
  table = "STBLDAT;0:[A:" + cycles + "," + res + "," + period.toFixed(0) +
":];";
  mips.SendCommand("MIPS-A", "STBLCLK,10500000\n")
```

```
  mips.SendCommand("MIPS-A", table)
  if (extTrig) {
    mips.SendCommand("MIPS-A", "STBLTRG,EXT\n")
    mips.SendCommand("MIPS-A", "SMOD,TBL\n")   // wait for external
trigger
  } else {
    mips.SendCommand("MIPS-A", "STBLTRG,SW\n")
    mips.SendCommand("MIPS-A", "SMOD,ONCE\n")
    mips.SendCommand("MIPS-A", "TBLSTRT\n")    // fire immediately
  }
EndScript

Script,abortA
  mips.SendCommand("MIPS-A", "TBLABRT\n")
EndScript
```

TBLABRT cancels any running table before a new one is downloaded. The clock is set to 10.5 MHz, so millisecond time values from the table are multiplied by 10500 to convert to clock ticks. If external trigger mode is checked the script enters SMOD,TBL and waits for a hardware trigger; otherwise it fires immediately with TBLSTRT.

> **Tip:** The generateB and abortB scripts are identical to generateA and abortA but reference Pulse generator B and send commands to MIPS-B instead of MIPS-A. Keeping the two generators independent means both can be configured and armed separately.

Appendix B: Resources

Install files and firmware:
https://drive.google.com/drive/folders/0B3IchPRNYqYIcjZhdGFVMVR1VzQ

Source code: https://github.com/GordonAnderson

JavaScript reference: https://www.w3schools.com/js/default.asp

Arduino Due USB driver: https://www.arduino.cc/en/Guide/ArduinoDue#toc10

Appendix C: Glossary

Item	Detail
ARB	Arbitrary Waveform Generator. Generates configurable waveforms for ion guide drive applications.

Item	Detail
cfg file	Plain text configuration file defining a custom control panel layout, widgets, and scripts.
Ccontrol	Generic control widget in a cfg file. The widget type (LineEdit, Button, CheckBox, ComboBox) determines behavior.
CV	Compensation Voltage. The DC voltage applied in FAIMS to select ions of a specific differential mobility.
DCbias	High-accuracy DC voltage outputs for ion guide and lens potentials.
Define	A cfg command that declares a named constant substituted into Ccontrol MIPSname fields at parse time, enabling portable panels.
DIO	Digital Input/Output. TTL-level signal lines for triggering, clocking, and status.
FAIMS	Field Asymmetric Ion Mobility Spectrometry. Separates ions using asymmetric electric fields.
GroupBox	A labeled border box grouping related controls. Supports an optional scrollable canvas mode.
mips object	The JavaScript object exposed to all control panel scripts providing UI access and hardware communication.
MIPS	Modular Instrument Platform System. A family of modular hardware controllers for scientific instruments.
PSG	Pulse Sequence Generator. Generates precisely timed sequences of digital and DC bias output changes.
RFCchannel	RF channel widget with full closed-loop control options. Compare with RFchannel (simplified, no closed-loop).
Script,Common	A special script block whose contents are appended to every other script, enabling shared utility functions.
Script,StartUp	A special script block executed automatically once when the control panel loads.
ScriptObj	A named alias associating a script object name with a script name.
SkipCount	Number of polling cycles to skip between automatic UI updates for a control.
Subroutine	A reusable layout template with named parameters, instantiated with the Call command.
TCPserver	A TCP listener opened by the control panel for remote automation from external applications.
Twave	Traveling Wave. A waveform technique that propagates a potential wave along an ion guide.