



Extron Electronics

INTERFACING, SWITCHING AND CONTROL

Extron Global Scriptor 设备模块使用指南

目录

概述	3
获取 Extron 设备模块和说明文档	3
在项目中添加 Extron GS 模块	3
在 Global Scriptor 代码中导入 Extron GS 模块	4
实例化创建 GS 模块通讯端口	5
串口通讯端口	5
以太网通讯端口 (TCP 和 UDP)	6
串口插入以太网通讯端口	7
HTTP 通讯端口	7
使用 Extron 模块	9
概述	9
设置模块的变量	9
设备控制的方法	10
Update 指令使用方法	10
SubscribeStatus 指令使用方法	11
ReadStatus 指令使用方法	12
方法的参数详解	13
Command 指令	13
Value	14
Qualifier (修饰符)	16
如何使用说明文档	17


设备连接状态 ConnectionStatus.....	18
其它示例	19
显示器的电源开关	19
音量状态电平更新	19
模块的多个实例	20
矩阵的切换指令	20
循环轮询 Polling Loop	21
TCP 连接处理	24
处理控制设备 TCP 连接意外中断:	25

概述

Extron 设计了专门用于串口和以太网的模块文件，可应用在 Global Scriptor 的项目中，使控制设备变得更加容易方便。该文档会讲述如何在 Global Scriptor 项目中导入模块文件，并利用其中的指令进行控制的详细步骤。

获取 Extron 设备模块和说明文档

在 Extron 网站的 “Global Scriptor Module” 下载页面中有可以提供的模块文件。

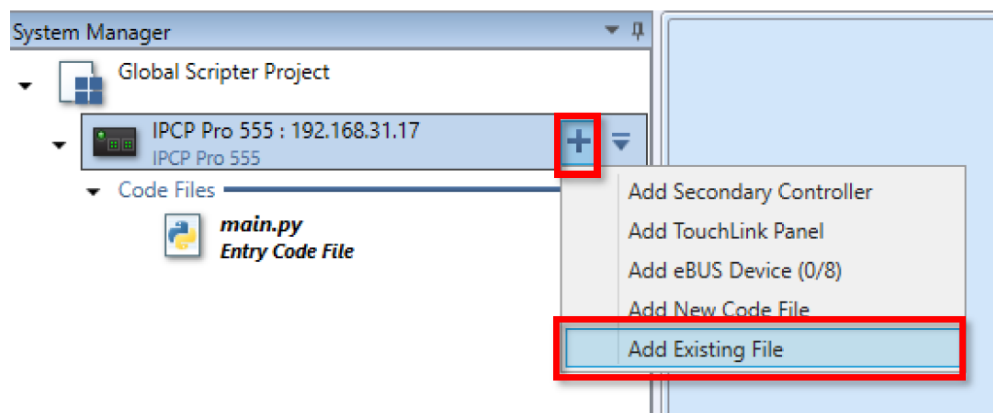
Model Number	Product Category	Version	Date Posted	Module	Communication Sheet
IN1604 HD	Scaler	1.4.1.2	Sep. 5, 2017	40 KB	 594 KB

每个模块含有一个 Python (.py) 的脚本文件，和一个 PDF 格式的说明文档，其中详细地说明了如何正确使用模块所需要的重要信息，包括：

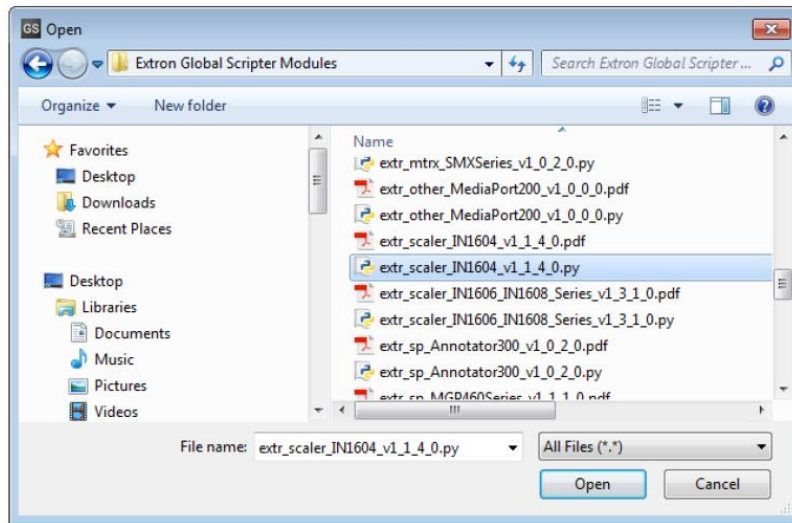
- 支持的设备型号
- 必须设置的任何特殊变量
- 支持的类(class)
- 支持的 Set 指令和其可能的值(values)和限定符(qualifiers)
- 支持的 Update 指令和其可能的值(values)和限定符(qualifiers)
- 串口线缆连接方式
- 以太网端口的连接信息
- 模块发送的指令字符串
- 关于模块和设备使用的其它信息

在项目中添加 Extron GS 模块

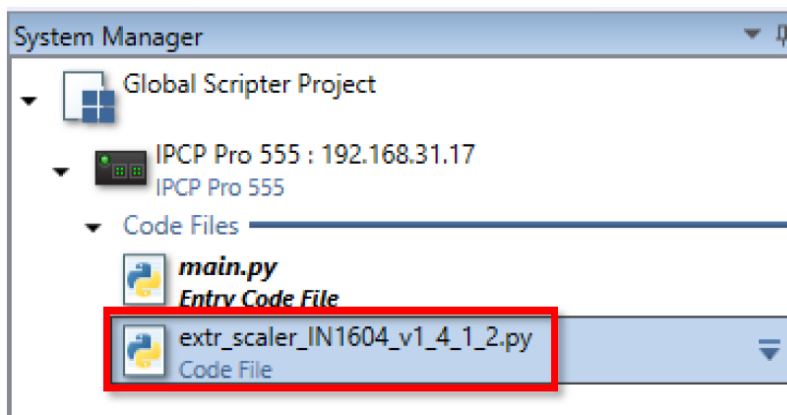
- 1、在 System manager 中，点击 Add Existing File。



- 2、在弹出菜单中选择相对应的文件，相应的文件请网站上进行下载。
(Extron GS 模块文件会持续更新，请使用最新版本的 GS 模块)



- 3、点击 Open 确定，将 Extron GS 的模块文件添加到项目文件中



在 Global Scripter 代码中导入 Extron GS 模块

在 Global Scripter 代码文件中应用 Extron GS 模块，需要在代码中使用 Python 关键字“import”进行定义。

在 **## Begin User Import ---** 和 **## End User Import ---** 的备注行中，根据相对应文件名称导入 Extron GS 模块。注意，不要写扩展名 .py。

import * Extron GS 模块文件名 * **as** * 模块别名 *

例如：

import extr_scaler_IN1604_v1_4_1_2 **as** IN1604Module

```

## Begin ControlScript Import -----
from extronlib import event, Version
from extronlib.device import eBUSDevice, ProcessorDevice, UIDevice
from extronlib.interface import (ContactInterface, DigitalIOInterface,
    EthernetClientInterface, EthernetServerInterfaceEx, FlexIOInterface,
    IRInterface, RelayInterface, SerialInterface, SWPowerInterface,
    VolumeInterface)
from extronlib.ui import Button, Knob, Label, Level
from extronlib.system import Clock, MESet, Wait

print(Version())

## End ControlScript Import -----
##
## Begin User Import -----
import extr_scaler_IN1604_v1_4_1_2 as IN1604Module
## End User Import -----
##
## Begin Device/Processor Definition -----

```

关键字 `as` 后面的名字是一个新的标识符用于指代导入的模块。在这个范例中，标识符 `IN1604Module` 就是引用的模块。

实例化创建 GS 模块通讯端口

Global Scriptor 代码文件中实例化创建串口或者以太网的方法，和实例化控制脚本的对象类似，控制端口的类型会在实例化端口时决定。通常在 **## Begin Communication Interface Definition ---** 和 **## End Communication Interface Definition ---** 的备注行之间创建实例化的代码。

Extron 模块支持类的数量由其支持的通讯协议决定，它们包括 Serial Class（串口类），SerialOverEthernetClass（以太网插入的串口），EtherClass（以太网类），HTTPClass（HTTP 类）。

可在说明文档中查阅：

- 支持的通讯协议
- 支持的类

串口通讯端口

通过 Extron 模块的 Serial Class（串口类）定义串口通讯端口，并设置正确的参数。这些参数和 Conrtro Script 的串口定义（Serial Interface）中的参数类似，只是多了型号（Model）参数，这个名称必须和受控设备型号相符。不需要单独定义此串口的参数。

可在说明文档中查阅:

- 支持的型号

Device Specifications

Device Type:	Audio Processor
Manufacturer:	Extron
Firmware Version:	1.00
Model(s):	DMP 128 Plus, DMP 128 Plus C, DMP 128 Plus C AT, DMP 128 Plus AT

Example code:

```
## Begin Communication Interface Definition -----  
DMP128 = DMPModule.SerialClass(IPCP, 'COM1', Baud=9600, Model='DMP 128 Plus')  
## End Communication Interface Definition -----
```

- 有效的串口设置参数

以太网通讯端口 (TCP 和 UDP)

通过 Extron 模块的 Ethernet Class (以太网类) 定义以太网通讯端口, 并设置正确的参数。这些参数和 Contro Script 的以太网客户端定义 (EthernetClientInterface) 中的参数类似, 只是多了型号 (Model) 参数, 这个名称必须和受控设备型号相符。不需要单独定义以太网端口的参数。

对于 TCP 设备, 创建的对象支持 connect 和 Disconnect 的方法, 这和 Control Script 中 EthernetClientInterface 是相同的。调用 Connect 初始化与受控设备的通讯, 在有需要的时候, 采用 Disconnect 方法断开与受控设备的通讯。

可在说明文档中查阅:

- 支持的型号
- IP 端口有效的设置参数
- 有效的 UDP 设备服务端口参数

Device Specifications

Device Type:	Audio Processor
Manufacturer:	Extron
Firmware Version:	1.00
Model(s):	DMP 128 Plus, DMP 128 Plus C, DMP 128 Plus C AT, DMP 128 Plus AT

示例代码（需要 Connect 进行 TCP 连接设备）：

```
## Begin Communication Interface Definition -----  
DMPEthernetPort1 = DMPModule.EthernetClass('192.168.254.249', 23, Model='DMP 128 Plus C AT')  
DMPEthernetPort1.Connect(timeout=5)  
## End Communication Interface Definition -----
```

示例代码（需要提供 UDP 设备的 ServicePort）：

```
## Begin Communication Interface Definition -----  
UDPCamera = CameraModule.EthernetClass('192.168.254.241', 52381, ServicePort=52380, Model='BRBK-IP10')  
## End Communication Interface Definition -----
```

串口插入以太网通讯端口

Extron 模块可能支持 SerialOverEthernetClass（串口插入以太网类），用于控制支持此功能的设备，参数设置和 Conrtro Script 的以太网客户端定义（EthernetClientInterface）中的参数类似，只是多了型号（Model）参数，这个名称必须和受控设备型号相符。不需要单独定义以太网端口的参数。

可在说明文档中查阅：

- 有效的型号名和参数
- 有效的串口设置信息

以下例子使用了 DTP CrossPoint 108 4K 矩阵，通过 PCS 软件先设置好以设备太网插入的端口信息。SerialOverEthernetClass 中设置的 Hostname 应该是 DTP Crosspoint 108 4K 矩阵的 IP 地址，端口号是指 PCS 中定义的插入端口号。

```
## Begin Communication Interface Definition -----  
SOEProjector = Projector.SerialOverEthernetClass('192.168.254.201', 2001, Model='SX800')  
## End Communication Interface Definition -----
```

HTTP 通讯端口

通过 Extron 模块的 HTTPClass（以太网类）定义以太网通讯端口，并设置正确的参数。这些参数包括 IP 地址，端口号，设备用户名，密码，和型号。除用户名和密码有可能不需要，其余所有参数都是必须的。如果不需要密码，密码可用 ' ' 替代。

可在说明文档中查阅：

- 有效的型号名和参数

- 有效的端口参数
- 用户名和密码默认值

以下圈出的即为支持的型号名称：

Device Specifications

Device Type:	Video Conference
Manufacturer:	LifeSize
Firmware Version:	N/A
Model(s):	Icon 600, Icon 800

代码示例： IP 192.168.254.250, HTTP 端口 80, 用户名 'admin' 密码 'extron'

```
## Begin Communication Interface Definition -----
VTC = DeviceModule.HTTPClass('192.168.254.240', 80, 'admin', 'extron', Model='Icon 600')
## End Communication Interface Definition -----
```

代码示例： 只输入用户名，不需要密码

```
## Begin Communication Interface Definition -----
VTC = DeviceModule.HTTPClass('192.168.254.240', 80, 'admin', '', Model='Icon 600')
## End Communication Interface Definition -----
```

代码示例： 不需要用户名和密码

```
## Begin Communication Interface Definition -----
VTC = DeviceModule.HTTPClass('192.168.254.240', 80, Model='Icon 600')
## End Communication Interface Definition -----
```


使用 Extron 模块

概述

使用 Extron 模块的类实例化并设置通讯端口时创建的对象支持一些方法 (Method)，可以建立和受控设备的通讯。使用这些方法 (Method)，你可以控制设备，也可以获取设备状态。使用 python 语言中点标的方式访问 (SerialClass, EthernetClass, HTTPClass, SerialOverEthernetClass) 中的方法。

重要注意事项：

- 可以通过手动 Update 的方法指令，根据程序的需求，进行状态轮询
- 对于 TCP 设备，涉及到设备如何连接或者断开连接来才能使用，需要确保在程序添加代码，处理好重新连接的逻辑
- 请详细阅读模块的说明文档，查看每个指令的用法和示例
- 有关 'command' 功能指令，'value' 数值 和 'qualifier' 标识符附加信息的用法，请查阅相关的参数部分

设置模块的变量

一些 Extron 的模块，需要设置变量后才能正常工作。常见的例子是需要填入设备的用户名和密码或者设备 ID 的信息。通常在实例化模块对象后，立即设置这些变量，以下的例子演示了设置这些变量的代码。

可在说明文档中查阅：

- 必须设置的变量
- 这些变量的有效值和类型

```
# Example of setting two instances of a module with different variables
cameraRoomA = VaddioModule.EthernetClass('192.168.254.249', 23, Model='ClearVIEW HD-USB PTZ')

cameraRoomA.deviceUsername = 'admin'
cameraRoomA.devicePassword = 'password_roomA'
cameraRoomA.DeviceID = '7'

cameraRoomB = VaddioModule.EthernetClass('192.168.254.248', 23, Model='ClearVIEW HD-USB PTZ')

cameraRoomB.deviceUsername = 'admin'
cameraRoomB.devicePassword = 'password_roomB'
cameraRoomB.DeviceID = '6'

# Once the variables are set, then proceed to Connect
cameraRoomA.Connect(timeout=5)
cameraRoomB.Connect(timeout=5)
```

设备控制的方法

Extron GS 模块提供了以下指令进行设备控制和设备状态获取：

- 设备控制： `Set`
- 设备状态获取： `Update`, `SubscribeStatus`, `ReadStatus`

Set指令使用方法 `Set (Command, Value, qualifier: None)`

Extron GS 模块可以使用 Set 指令发送代码，实现设备的控制。

参考模块的说明文件，决定是否需要修饰符 (Qualifier) ，以及修饰符的值

Parameters:

command (*string*) – 指令的名称

value (*string, int, float, None*) – 针对设备设定的值

qualifier (*dict, None*) – 用字典形式标注的命令参数 (如果需要)

例如：

```
# Examples
extronScaler1.Set('Input', '4')
extronScaler1.Set('AspectRatio', 'Fill Mode', {'Input' : '2'})
```

Update指令使用方法 `Update(Command,qualifier=None)`

向受控设备发送查询指令。这个方法不会返回一个值。需使用 `ReadStatus` 或 `SubscribeStatus` 方法去利用状态的信息。

参考模块的说明文件，决定是否需要修饰符 (Qualifier) ，以及修饰符的值

Parameters:

command (*string*) – 指令名称

qualifier (*dict, None*) –用字典形式标注的命令参数 (如果需要)

例子：

```
# Examples
extronScaler1.Update('Input')
extronScaler1.Update('AspectRatio', {'Input' : '2'})
```

SubscribeStatus指令使用方法 SubscribeStatus(Command,qualifier,callback)

当模块中设备的状态发生更新时，触发用户所自定义的函数。

如果指定了修饰符（qualifier），只有当这个特定的修饰符对应状态发生改变时，才会调用 Callback 的函数。如果修饰符（qualifier）被设为 None，任何修饰符对应状态发生改变时都将调用 callback 函数。以上两种情况，修饰符都会被传递 callback 函数中。

Parameters:

command (*string*) – 需要订阅的功能指令名称

qualifier (*dict, None*) – 以字典或None形式命名的参数，和其它方法不同，这里需要用修饰符是必须要填写的。

callback (*function*) – 当状态更新时，调用的自定义函数，其自定义函数需要接收三个参数

SubscribeStatus 将 Extron 模块新获取的状态和程序员创建的函数绑定。只有检测到订阅的状态有更新时，包括最初收到的状态，才会调用 Callback 函数。

有些反馈可能是设备主动发出的，在这种情况下，不需要持续调用 Update，SubscribeStatus 也能触发 callback 函数。建议在获取初始状态后立即调用 Update。请参考模块的说明文档，确定这是否和你需要控制的设备情况一样。

通常，执行任何 Update 之前，先设置好所有期望的 SubscribeStatus 的功能。

创建的回调自定义函数，必须带有三个参量（参考模块的说明文档查询指定命令的期望数值，修饰符和类型）。Extron 模块会调用你指定的 Callback 函数，并传递相应的参量 (Argument)：

1. 第一个参数是命令的名称 command，模块返回字符串信息
2. 第二个参数是新接收到的状态值 value，模块返回字符串，整数或浮点数的信息
3. 第三个参数是有更新状态的指令对应的修饰符，模块返回 None 或字典信息，如果这条指令包含一个修饰符。

```

# Examples
mesInputButtons = MESet([btnInput1, btnInput2, btnInput3, btnInput4])

def ReceivedNewInputStatus(command, value, qualifier):

    # Set the currently selected button on an MESet of Buttons
    # based on the new value

    if value == '1':
        mesInputButtons.SetCurrent(btnInput1)
    elif value == '2':
        mesInputButtons.SetCurrent(btnInput2)
    elif value == '3':
        mesInputButtons.SetCurrent(btnInput3)
    else:
        mesInputButtons.SetCurrent(btnInput4)

extronScaler1.SubscribeStatus('Input', None, ReceivedNewInputStatus)

def ReceivedNewAspectRatioStatus(command, value, qualifier):

    # For commands that have a qualifier, the qualifier dictionary that has
    # the new status is passed into the "qualifier" argument and can be used
    # to determine the exact status that was changed

    affectedInput = qualifier['Input']

    if affectedInput == '1':
        print('New Aspect Ratio for Input 1 is', value)
    elif affectedInput == '2':
        print('New Aspect Ratio for Input 2 is', value)
    elif affectedInput == '3':
        print('New Aspect Ratio for Input 3 is', value)
    else:
        print('New Aspect Ratio for Input 4 is', value)

extronScaler1.SubscribeStatus('AspectRatio', None, ReceivedNewAspectRatioStatus)

```

ReadStatus 指令使用方法 *ReadStatus(command, qualifier=None)*

ReadStatus 可以返回最新存储在模块中特定指令和修饰符组合的状态值。**它不会向设备发送查询指令。**只是使用 Update 方法进行查询。

可参考模块的说明文档确定是否需要修饰符，以及期望的反馈类型。

如果还没有状态存储，ReadStatus 将返回 None 。例如，还没有针对特定的指令发出 Update 时。

ReadStatus 也可能返回一个和实际状态不同的值。例如设备状态的变化发生在最近一次 Update 之后。

Extron Global Scripser 代码可以使用 ReadStatus 指令，检索保存在 GS 模块的设备状态信息。

	command (<i>string</i>) – 功能指令名称
参数:	qualifier (<i>dict, None</i>) – 用字典标识的命令参数(如果需要)
返回:	最新的状态数值
返回类型:	string, int, float, None

```
# Examples
extronScaler1.Update('Input')
extronScaler1.Update('AspectRatio', {'Input' : '2'})

# A small time delay is introduced to allow the communication between the module
# and device to occur.
@Wait(1.5)
def PrintStatus():
    currentInputStatus = extronScaler1.ReadStatus('Input')
    print('The switcher is currently on Input', currentInputStatus)

    currentAspectRatioStatus2 = extronScaler1.ReadStatus('AspectRatio', {'Input' : '2'})
    print('The aspect ratio of Input 2 is currently set to', currentAspectRatioStatus2)
```

方法的参数详解

本章节会详细地介绍Set, Update, ReadStatus 和 SubscribeStatus 中使用的参数，其中包括 command 命令、value 值和 qualifier 标识符。

Command 指令

Command 指令是设备要控制的功能名称，比如说电源、输入、视频静音、音量等。

设备所支持的 Command 指令，可以通过 Extron 模块说明文档中的 Set Commands 和 Status Available 列表中找到，Set Command 表格中的指令可用于控制设备，可被 Set 方法所使用。Status Available 表格中的指令是来自这个设备的状态，可通过 Update，ReadStatus，和 SubscribeStatus 方法所用，（有些指令可能不全部支持这 3 种方法，详情请参考模块说明文档）。你代码中所有的指令名称必须是字符串。

以下的 Input 输入, MenuNavigation 菜单导航, OnScreenDisplay 在屏显示, Power 开关, 就是设备所支持的功能性指令。

Command	Value	Value	Value
Input	'DTV (Antenna)'	'DTV (Cable)'	'Analog (Antenna)'
	'Analog (Cable)'	'AV 1'	'AV 2'
	'Component 1'	'Component 2'	'RGB-PC'
	'HDMI 1'	'HDMI 2'	'HDMI 3'
	'HDMI 4'		
# Input example InterfaceName.Set('Input', 'DTV (Antenna)')			
Command	Value	Value	Value
MenuNavigation	'Menu'	'Up'	'Down'
	'Left'	'Right'	'Enter'
	'Exit'	'Back'	
# MenuNavigation example InterfaceName.Set('MenuNavigation', 'Menu')			
Command	Value	Value	
OnScreenDisplay	'On'	'Off'	
# OnScreenDisplay example InterfaceName.Set('OnScreenDisplay', 'On')			
Command	Value	Value	
Power	'On'	'Off'	
# Power example InterfaceName.Set('Power', 'On')			

Value

Value 是您想更改的设备功能命令的对应状态。例如, Power 命令, 其 Value 值是有 'On' 和 'Off' ; Volume 音量, 其 Value 值是 1、2、3、4 到 100。

每条指令对应的值都在 Extron 模块的说明文档中详细列出。

Command Input	Value 'DTV (Antenna)' 'Analog (Cable)' 'Component 1' 'HDMI 1' 'HDMI 4'	Value 'DTV (Cable)' 'AV 1' 'Component 2' 'HDMI 2'	Value 'Analog (Antenna)' 'AV 2' 'RGB-PC' 'HDMI 3'
# Input example InterfaceName.Set('Input', 'DTV (Antenna)')			
Command MenuNavigation	Value 'Menu' 'Left' 'Exit'	Value 'Up' 'Right' 'Back'	Value 'Down' 'Enter'
# MenuNavigation example InterfaceName.Set('MenuNavigation', 'Menu')			
Command OnScreenDisplay	Value 'On'	Value 'Off'	
# OnScreenDisplay example InterfaceName.Set('OnScreenDisplay', 'On')			
Command Power	Value 'On'	Value 'Off'	
# Power example InterfaceName.Set('Power', 'On')			

在程序的代码中，其 Value 是以下 4 种 Python 类型数据：

String (字符串)

字符串在说明文档中用单引号表示。

Command Power	Value 'On'	Value 'Off'
# Power example InterfaceName.Set('Power', 'On')		

Integer (整数)

如果该数值以范围来表示，并且步进值是整数，则该值必须是整数类型。

Command Volume	Value 0 to 100 in steps of 1
# Volume example InterfaceName.Set('Volume', 100)	

Float (浮点数)

如果该数值以范围来表示，并且步进值是小数，则该值必须是浮点类型。

Command	Value
Volume	0 to 100 in steps of 0.1
<pre># Volume example InterfaceName.Set('Volume', 0.1)</pre>	

None

如果该 value 值被列为 None，则该值必须为 None。

Command	Value
AutoImage	None
<pre># AutoImage example InterfaceName.Set('AutoImage', None)</pre>	

Qualifier（修饰符）

Qualifier 修饰，进一步指定了指定命令的确切控制点。例如，对于输入通道静音功能，其对应可以开和关，第三个参数 Qualifier（修饰符）就可以指定是哪一个输入通道进行静音。

Qualifier 的格式为 Python 字典格式。在字典的架构中，key 是对应索引的名称，其 value 值是对应 key 的存储的数值。

设备代码的说明文档中，列明了其修饰符的键(Key)值和其对应的 Value 值。

Qualifier 的 Value 值是三种类型之一：字符串、整数或浮点。

以下的文档例子，列出其虚拟的返回通道，其通道可以从 A 到 P 之间进行选择，然后进行静音或取消静音。

Command	Value	Value	
VirtualReturnMute	'On'	'Off'	
Qualifier Key	Qualifier Value	Qualifier Value	Qualifier Value
'Channel'	'A'	'B'	'C'
	'D'	'E'	'F'
	'G'	'H'	'I'
	'J'	'K'	'L'
	'M'	'N'	'O'
	'P'		
<pre># VirtualReturnMute example InterfaceName.Set('VirtualReturnMute', 'On', {'Channel': 'A'})</pre>			

以下是使用变量存储字典类型修饰符的范例，key 是 'Channel'，其对应的 Value 是 'A'。


```
qualifierDict = {'Channel' : 'A'}
DMP128.Set('VirtualReturnMute', 'On', qualifierDict)
```

以下的例子是 Extron 矩阵的切换指令，采用字典存储多个修饰符 Key 和修饰符对应的数值（矩阵的输入和输出的通道，以及切换类型）。

```
# Perform an Audio/Video tie from Input 1 to Output 3
qualifierDict = {
    'Input' : '1',
    'Output' : '3',
    'Tie Type': 'Audio/Video',
}

matrixDTP108.Set('MatrixTieCommand', None, qualifierDict)
```

相应 Update，其修饰符 qualifier 参数，也是用字典标示：

```
qualifierDict = {
    'Output' : '8',
    'L/R' : 'Left',
}

matrixDTP108.Update('DTPOutputLevel', qualifierDict)
```

如何使用说明文档

设备代码的说明文档中，会列明了该设备所支持的 Set Command 功能。

Command	Value	Value	Value
3DMode	'3D On'	'3D Off'	'3D to 2D'
	'2D to 3D'		
Qualifier Key	Qualifier Value	Qualifier Value	Qualifier Value
'Option'	'Top and Bottom'	'Side by Side'	'Check Board'
	'Frame Sequential'		
Qualifier Key	Qualifier Value	Qualifier Value	
'Direction'	'Right to Left'	'Left to Right'	
Qualifier Key	Qualifier Value		
'3D Depth'	0 to 20 in steps of 1		
# 3DMode example InterfaceName.Set('3DMode', '3D On', {'Option': 'Top and Bottom', 'Direction': 'Right to Left', '3D Depth': 20})			
Command	Value	Value	Value
AspectRatio	'4:3'	'16:9'	'Set by Program'
	'Just Scan'	'Cinema Zoom 1'	'Cinema Zoom 2'
	'Cinema Zoom 3'	'Cinema Zoom 4'	'Cinema Zoom 5'
	'Cinema Zoom 6'	'Cinema Zoom 7'	'Cinema Zoom 8'
	'Cinema Zoom 9'	'Cinema Zoom 10'	'Cinema Zoom 11'
	'Cinema Zoom 12'	'Cinema Zoom 13'	'Cinema Zoom 14'
	'Cinema Zoom 15'	'Cinema Zoom 16'	
# AspectRatio example InterfaceName.Set('AspectRatio', '4:3')			

Status Available 列表，列出了“可用 Update”的指令：

Command	Value	Value	Value
AspectRatio	'4:3'	'16:9'	'Zoom'
	'Set by Program'	'Just Scan'	'Cinema Zoom 1'
	'Cinema Zoom 2'	'Cinema Zoom 3'	'Cinema Zoom 4'
	'Cinema Zoom 5'	'Cinema Zoom 6'	'Cinema Zoom 7'
	'Cinema Zoom 8'	'Cinema Zoom 9'	'Cinema Zoom 10'
	'Cinema Zoom 11'	'Cinema Zoom 12'	'Cinema Zoom 13'
	'Cinema Zoom 14'	'Cinema Zoom 15'	'Cinema Zoom 16'
<pre># AspectRatio examples InterfaceName.Update('AspectRatio') Value = InterfaceName.ReadStatus('AspectRatio') InterfaceName.SubscribeStatus('AspectRatio', None, FeedbackHandler)</pre>			
Command	Value	Value	
AudioMute	'On'	'Off'	
<pre># AudioMute examples InterfaceName.Update('AudioMute') Value = InterfaceName.ReadStatus('AudioMute') InterfaceName.SubscribeStatus('AudioMute', None, FeedbackHandler)</pre>			
Command	Value	Value	
ConnectionStatus	'Connected'	'Disconnected'	
<pre># ConnectionStatus examples Value = InterfaceName.ReadStatus('ConnectionStatus') InterfaceName.SubscribeStatus('ConnectionStatus', None, FeedbackHandler)</pre>			

“Value” 部分中列出的状态，会传递到 SubscribeStatus 中使用的自定义 Callback 函数中，它们也可能是 ReadStatus 指令的返回值。但该 Value 不用于 Update 指令。

在极少数情况下，Status Available 表中的命令只支持 Update，而不支持 ReadStatus 和 Subscribe。同样，有少数指令支持 ReadStatus 和 Subscribe，但不支持 Update。欲知详情请查阅设备模块的说明文档。

设备连接状态 **ConnectionStatus**

连接状态是指 Extron 中控和受控设备之间的通讯状态，Extron 的代码库会对设备的连接状态自动进行更新，其注意事项如下：

- **ConnectionStatus** 连接状态，是所有 Extron 模块中都会有的特殊状态变量，确保了至少有一个状态可用
- 连接状态的可能状态有：连接 ‘Connected’ 和 ‘Disconnected’ 断开
- 在第一次尝试更新之后，ConnectionStatus 将变成 “Connected”
- 如果连续 15 次 Update 更新调用之后，都没有收到受控设备的响应，将变成断开连接 ‘Disconnected’，该数值可以通过修改代码中的 connectionCounter 的变量值，详细请参阅设备模块的说明文档。
- 如果 Update 指令成功接收到设备的返回信息，连接状态将再次变为 “Connected”

其它示例

显示器的电源开关

下面是创建 Power On 开机和 Power Off 关机按钮的示例。

可以通过 MESet 中设置两个按键互斥，设置按钮的 Pressed 按下事件，然后调用 Extron 设备模块的 Set 指令，进行电源的开和关，并得到按钮的反馈。其中 MESet，常用于用户控制界面 UI 上的视觉反馈，并不会影响 Extron 设备模块的代码行动。

```
## Begin User Import -----
import lg_display_xxLW_xxLZ_xxPZ_xxLV_xxLK_v1_0_0_0 as DisplayModule
## End User Import -----
##
## Begin Device/Processor Definition -----
IPCP = ProcessorDevice('IPCP550')
## End Device/Processor Definition -----
##
## Begin Device/User Interface Definition -----
TLP = UIDevice('TLP1022')
## End Device/User Interface Definition -----
##
## Begin Communication Interface Definition -----
display1 = DisplayModule.SerialClass(IPCP, 'COM1', Baud=9600, Model='47LW7700')
## End Communication Interface Definition -----

btnPowerOn = Button(TLP, 1)
btnPowerOff = Button(TLP, 2)
mesPowerButtons = MESet([btnPowerOn, btnPowerOff])
## Event Definitions -----
@event([btnPowerOn, btnPowerOff], 'Pressed')
def PowerButtonPressed(button, state):
    if button is btnPowerOn:
        display1.Set('Power', 'On')

    elif button is btnPowerOff:
        display1.Set('Power', 'Off')

    mesPowerButtons.SetCurrent(button)
## End Events Definitions-----
```

音量状态电平更新

以下是创建并跟踪音量电平表状态的示例。SubscribeStatus 用于在每次模块接收到 Volume 的不同状态时，设置音量的级别。Timer 定时器用于定期进行音量的更新。有关创建轮询循环的更多示例，请参阅轮询循环部分。

```

lvlVolumeStatus = Level(TLP, 3)
lvlVolumeStatus.SetRange(0, 100)

def ReceivedNewVolumeStatus(command, value, qualifier):
    lvlVolumeStatus.SetLevel(value)

display1.SubscribeStatus('Volume', None, ReceivedNewVolumeStatus)

def PollVolume(timer, count):
    display1.Update('Volume')

volumePollTimer = Timer(3, PollVolume)

```

模块的多个实例

既然设备模块中使用了多个类，所以一个模块可以创建多个实例，并在 GS 项目中使用。下面的例子是通过使用相同的 Extron 模块，实际控制 3 个矩阵设备，其中 2 个是通过串口控制，1 个是通过以太网控制。

```

import extr_matrix_XTPIICrossPointSeries_v1_1_1_1 as moduleXTP

matrixRoomA = moduleXTP.SerialClass(IPCP, 'COM1', Baud=9600, Model='XTP II CrossPoint 1600')
matrixRoomB = moduleXTP.SerialClass(IPCP, 'COM2', Baud=9600, Model='XTP II CrossPoint 1600')

matrixMainHall = moduleXTP.EthernetClass('192.168.254.230', 23, Model='XTP II CrossPoint 6400')
matrixMainHall.devicePassword = 'extron'
matrixMainHall.Connect(timeout=5)

# Recall Preset 1 on all matrix switchers
matrixRoomA.Set('PresetRecall', '1')
matrixRoomB.Set('PresetRecall', '1')
matrixMainHall.Set('PresetRecall', '1')

```

矩阵的切换指令

所有的 Extron 的矩阵设备，通过调用 MatrixTieCommand 指令，进行输入和输出的视频和音频的切换操作。

指令格式是：Value 值是 None, 通过修饰符 Qualifier 指定输入通道，输出通道和切换类型。如果输入通道值为 0，则切断连接。

Command	Value		
MatrixTieCommand	None		
Qualifier Key	Qualifier Value		
'Input'	'0' – '10'		
Qualifier Key	Qualifier Value	Qualifier Value	
'Output'	'1' – '8'	'All'	
Qualifier Key	Qualifier Value	Qualifier Value	Qualifier Value
'Tie Type'	'Audio'	'Audio/Video'	'Video'
# MatrixTieCommand example InterfaceName.Set('MatrixTieCommand', None, {'Input': '0', 'Output': '1', 'Tie Type': 'Audio'})			

以下示例代码实现按下按钮后，动态创建字典，并根据按下哪个按钮来调整字典内容，最终实现了矩阵的切换

```
@event([btnLaptopToLeft, btnLaptopToRight, btnLaptopToBoth], 'Pressed')
def MakeLaptopMatrixTies(button, state):

    qualifierDict = {
        # The laptop source is on input 3 of the matrix
        'Input'      : '3',
        'Tie Type'   : 'Audio/Video',
    }

    if button is btnLaptopToLeft:

        # Adding the Output Qualifier Key to qualifierDict
        # and set its Qualifier Value to '1'
        qualifierDict['Output'] = '1'

        # This will route Input 3 to Output 1 for Audio/Video
        matrixDTP108.Set('MatrixTieCommand', None, qualifierDict)

    elif button is btnLaptopToRight:

        # Adding the Output Qualifier Key to qualifierDict
        # and set its Qualifier Value to '2'
        qualifierDict['Output'] = '2'

        # This will route Input 3 to Output 2 for Audio/Video
        matrixDTP108.Set('MatrixTieCommand', None, qualifierDict)

    elif button is btnLaptopToBoth:

        # Need to do the same exercise twice since the desired functionality
        # is the laptop source is routed to both the left and right outputs

        # This will route Input 3 to Output 1 for Audio/Video
        qualifierDict['Output'] = '1'
        matrixDTP108.Set('MatrixTieCommand', None, qualifierDict)

        # This will route Input 3 to Output 2 for Audio/Video
        qualifierDict['Output'] = '2'
        matrixDTP108.Set('MatrixTieCommand', None, qualifierDict)
```

循环轮询 Polling Loop

可使用 ControlScript 的 Wait 类，并结合轮询循环的方法，连续调用 Extron Module 的 Update 方法更新设备的状态，下面列举了几种不同的方法。

- 使用单个 Wait 对象、查询列表，以及一个 Index 变量控制发出的查询指令内容：

示例代码：

```
# Example of continuously calling Update to query three different statuses
# with 0.3 seconds of delay in between each query.
MATRIX_QUERY_DELAY = 0.3

MATRIX_QUERY_LIST = [
    ('ExecutiveMode', None),
    ('VideoMute', {'Output' : '1'}),
    ('VideoMute', {'Output' : '2'}),
]

queryIndex = 0

def QueryDTP108():
    global queryIndex
    command, qualifier = MATRIX_QUERY_LIST[queryIndex]
    matrixDTP108.Update(command, qualifier)

    queryIndex += 1
    if queryIndex >= len(MATRIX_QUERY_LIST):
        queryIndex = 0

    matrixPollingWait.Restart()

matrixPollingWait = Wait(MATRIX_QUERY_DELAY, QueryDTP108)
```

如果需要添加更多的查询指令，在 MATRIX_QUERY_LIST 列表中追加数据元组（指令 command，修饰符 qualifier）。如果不需要修饰符，则写 None。

```
MATRIX_QUERY_LIST = [
    ('ExecutiveMode', None),
    ('VideoMute', {'Output' : '1'}),
    ('VideoMute', {'Output' : '2'}),
    ('InputSignalStatus', {'Input' : '1'}),
    ('Freeze', {'Output' : '5'}),
]
```

- 与前面的示例类似，但是使用 Python 的内置 `itertools.cycle` 功能:

```
# Example of continuously calling Update to query three different statuses
# with 0.3 seconds of delay in between each query.
import itertools

MATRIX_QUERY_DELAY = 0.3

MATRIX_QUERY_LIST = [
    ('ExecutiveMode', None),
    ('VideoMute', {'Output' : '1'}),
    ('VideoMute', {'Output' : '2'}),
]

matrixCycle = itertools.cycle(MATRIX_QUERY_LIST)

def QueryDTP108():
    command, qualifier = next(matrixCycle)
    matrixDTP108.Update(command, qualifier)
    matrixPollingWait.Restart()

matrixPollingWait = Wait(MATRIX_QUERY_DELAY, QueryDTP108)
```

- 与前面的示例类似，但是使用 Python 的内置 `collections.deque` 功能:

```
# Example of continuously calling Update to query three different statuses
# with 0.3 seconds of delay in between each query.
import collections

MATRIX_QUERY_DELAY = 0.3

MATRIX_QUERY_LIST = [
    ('ExecutiveMode', None),
    ('VideoMute', {'Output' : '1'}),
    ('VideoMute', {'Output' : '2'}),
]

matrixQueue = collections.deque(MATRIX_QUERY_LIST)

def QueryDTP108():
    matrixDTP108.Update(*matrixQueue[0])
    matrixQueue.rotate(-1)
    matrixPollingWait.Restart()

matrixPollingWait = Wait(MATRIX_QUERY_DELAY, QueryDTP108)
```

TCP 连接处理

在程序的代码中必须处理 Extron 设备模块的 TCP 物理连接。这包括初次尝试建立 TCP 连接和 TCP 断线重连时调用的 Connect 方法。下面的示例使用模块的 ConnectionStatus 以及 Connected 事件，来确定 TCP 连接状态是正常还是断开。

```
matrixDTP108 = ExtronModule.EthernetClass('192.168.254.202', 23, Model='DTP Crosspoint 108 4K')

# Handling TCP connection of the Extron Matrix
def AttemptConnectMatrix():
    """Attempt to create a TCP connection to the Matrix.
    If it fails, retry in 15 seconds.
    """
    print('Attempting to connect to the Matrix')
    result = matrixDTP108.Connect(timeout=5)
    if result != 'Connected':
        reconnectWait.Restart()

reconnectWait = Wait(15, AttemptConnectMatrix)

def ReceivedMatrixConnectionStatus(command, value, qualifier):
    """If the module's ConnectionStatus becomes Disconnected, then many
    consecutive Updates have failed to receive a response from the device.
    Attempt to re-establish the TCP connection to the Matrix by calling
    Disconnect on the module instance and restarting reconnectWait.
    """
    print('Matrix module ConnectionStatus is', value)
    if value == 'Disconnected':
        matrixDTP108.Disconnect()
        reconnectWait.Restart()

matrixDTP108.SubscribeStatus('ConnectionStatus', None, ReceivedMatrixConnectionStatus)

@event(matrixDTP108, 'Connected')
def MatrixPhysicalConnectionEvent(interface, state):
    """If the TCP connection has been established physically, stop attempting
    reconnects. This can be triggered by the initial TCP connect attempt in
    the Initialize function or from the connection attempts from
    AttemptConnectMatrix.
    """
    reconnectWait.Cancel()

def Initialize():
    matrixDTP108.Connect(timeout=5)

Initialize()
```

当收到“断开连接”状态时，您可以将正在处理轮询循环的 Wait 对象上的 Cancel，进行停止轮询。在成功尝试 Connect 之后，调用 Wait 对象的 Restart 以恢复轮询。

当重连后进行 Restart：


```
# Handling TCP connection of the Extron Matrix
def AttemptConnectMatrix():
    """Attempt to create a TCP connection to the Matrix.
    If it fails, retry in 15 seconds.
    """
    print('Attempting to connect to the Matrix')
    result = matrixDTP108.Connect(timeout=5)
    if result != 'Connected':
        reconnectWait.Restart()
    else:
        matrixPollingWait.Restart()
```

当模块断开连接时调用 Cancel 取消：

```
def ReceivedMatrixConnectionStatus(command, value, qualifier):
    """If the module's ConnectionStatus becomes Disconnected, then many
    consecutive Updates have failed to receive a response from the device.
    Attempt to re-establish the TCP connection to the Matrix by calling
    Disconnect on the module instance and restarting reconnectWait.
    """
    print('Matrix module ConnectionStatus is', value)
    if value == 'Disconnected':
        matrixDTP108.Disconnect()
        reconnectWait.Restart()
        matrixPollingWait.Cancel()
```

处理控制设备 TCP 连接意外中断

在少数情况下，调用 Set 或 Update 指令，可能会产生 Python 的 BrokenPipeError 异常，这是由于受控设备没有正确的断开 TCP socket 接口，为了处理此错误，可以使用 try / exception 的语句。

```
try:
    dvProjector.Update('Power')
except BrokenPipeError:
    dvProjector.Disconnect()
```