Extron

What's New in Python 3.11?

Exton has updated the version of Python used in our current models of Control Processors to Python 3.11. The Python Upgrade will first be introduced in the IPCP Pro xi and IPCP Pro Q xi firmware and will follow shortly thereafter in our firmware for the TLP Pro Touchpanels with LinkLicense, VCA/VCP virtual control processors and WC Pro workspace controllers. Control processors not listed above will continue to use Python 3.5.

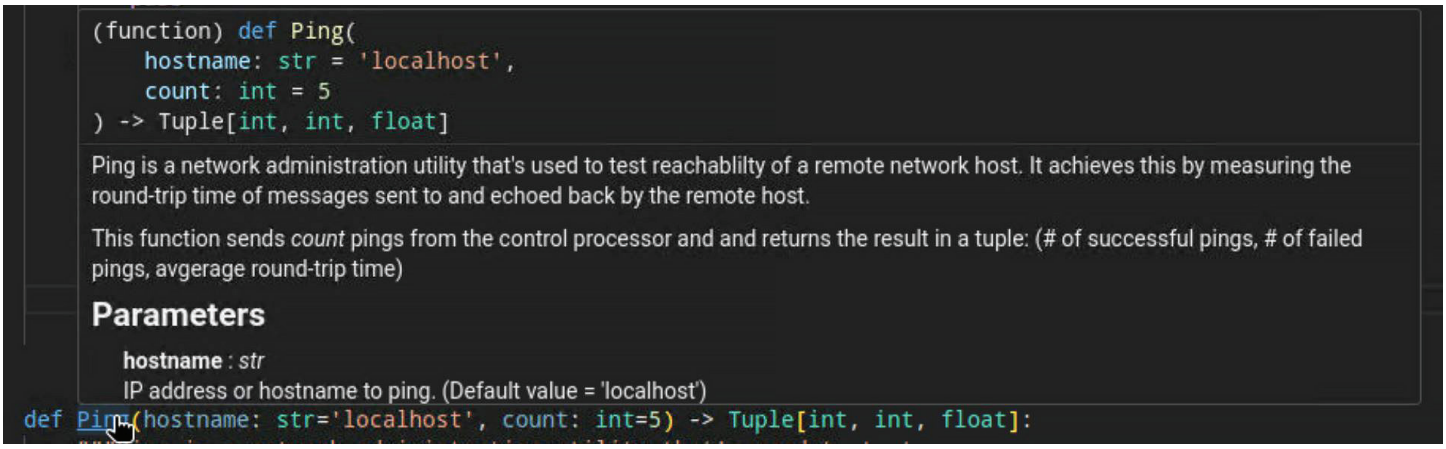
Please refer to the Release Notes for your processor's current firmware to see if the firmware supports Python 3.11.

Performance Upgrades

Between Python 3.5 and Python 3.11, Python has undergone several major upgrades that have resulted in significant improvements. Much of the performance of Extron configurations and programs is determined by the bidirectional communications between devices. As a result, it may be difficult to perceive the overall performance gains within the system.

Type Hinting

Type hinting was introduced in Python 3.5 and has been improved in subsequent versions. Type hints are helpers that let the user know what data types methods and classes are expected to be passed to them and, likewise, what data type you should expect to be returned by them. It is not type declaration – like in C/C++ and other derivative/similar languages – where the compiler is guaranteeing the data type. The ControlScript Extension for VS Code now includes complete type-hinting of extronlib objects to provide a better coding experience.



A new feature adds the ability to provide type hinting for a variable at the same time as the value of the variable is assigned.

```
16
17   Sources: list[str] = ["PC", "Laptop HDMI", "Laptop USB-C", "Media Player"]
18
```

f-strings – Formatted Strings Literals

f-strings, aka formatted string literals, put the variables and formatting into the string definition itself. This is considered a more readable alternative to placing the formatting in the Format method as in Python 3.5. f-strings have become popular within the Python community, but caution should be used when developing code or modules that may be used on older processors or firmware running Python 3.5.

```
ActiveSource = 'Laptop1'

# String formatting using format method.
print("ActiveSource = '{}'".format(ActiveSource))
# Output in Trace: ActiveSource = 'Laptop1'

# String formatting using f-strings (Formatted Literal Strings)
# written similar to the format method shown above.
print(f"ActiveSource = '{ActiveSource}'")
# Output in Trace: ActiveSource = 'Laptop1'

# String formatting using the f-string self-documenting expression.
print(f'{ActiveSource = }')
# Output in Trace: ActiveSource = 'Laptop1'


```

String Methods for Removing Prefixes / Suffixes

It's now easier than ever to remove unwanted portions of strings, whether returned from a device in a ReceiveData event or when reading a file. str.removeprefix() and str.removesuffix() will remove the string specified as the argument if it is found in the string. These two methods can be used sequentially in the same expression to remove both a prefix and suffix.

```
# Response from Extron NetPA U 1004 Set Gain Command where the object ID
# is between 40100 - 40103 for inputs 1-4
respString = 'DsG40100*-4.7\r\n'
respString = respString.removeprefix('DsG4010').removesuffix('\r\n')
# respString is now: '0*-4.7'

chan, value = respString.split('*')
# chan = '0'
# value = '-4.7'
```

Underscore

Single underscore characters can now be used as a digit group separator for long numbers to improve readability; however leading, trailing, or multiple underscore characters in a row are not allowed. This can be used with any numeric type and can make reading very large numbers and hexadecimal numbers easier. Notice in the examples that internally, the number is represented without the underscore.

```
>>> DIST_SUN_AVG_KM = 149_600_000
>>> DIST_SUN_AVG_KM
149600000
>>> DIST_MOON_AVG_KM = 384_400
>>> DIST_MOON_AVG_KM
384400
>>> BACNET_PORT_NUM = 0x_BA_C0
>>> BACNET_PORT_NUM
47808
```

match/case - Structural Pattern Matching

match/case is an alternative conditional statement to the if/elif/else statements. The value of the variable passed in the match statement is evaluated against the values in the case statements in the or that they appear. The underscore is evaluated as a default condition if no other case statements evaluate as true. This structure can be extended easily if additional conditions or patterns are required later.

```
 82  def SetButtonColor(button: Button):
 83      match var.TieType:
 84          case '$':
 85              button.SetState(2)
 86          case '%':
 87              button.SetState(3)
 88          case '!':
 89              button.SetState(4)
 90          case _:
 91              print('TieType is not a valid value', var.TieType)
 92
 93
 94  # btns: Tie Types; Select Tie Type; Set Current Tie Type label
 95  @eventEx([btnAudio, btnVideo, btnBoth], 'Pressed')
 96  def btnTieTypeEvent(button: Button, state: str):
 97      TypeGroup.SetCurrent(button)
 98      match button.Name:
 99          case 'AudioTie':
100              var.TieType = '$'
101          case 'VideoTie':
102              var.TieType = '%'
103          case 'AVTie':
104              var.TieType = '!'
105          case _:
106              print('This should not ever happen')
```

Walrus Operator (:=) – Assignment Expressions

Assignment expressions combine an assignment with an expression evaluation into one operation. This can often be less readable than the traditional way of writing it, so use sparingly. Even the Python docs say, “Try to limit use of the walrus operator to clean cases that reduce complexity and improve readability.” In the example below an assignment expression is used on line 62 to combine the operations shown on lines 45 and 48.

```
42  # NOTE: Create a RegEx object that holds the info returned from the search
43  # method using ResponseString.
44
45  MatchObject = ResponsePattern.search(ResponseString)
46
47  # If MatchObject matches the pattern then parse each group.
48  if MatchObject:
49      Output = MatchObject.group(1)
50      Input = MatchObject.group(2)
51      Mode = MatchObject.group(3)
52
53      print('Input: {0}, Output: {1}, Type: {2}'.format(Input, Output, Mode))
54      # prints the info to trace messages
55
56  # NOTE: Using the Assignment expression, called the Walrus Operator,
57  # combines this into a single statement.
58
59  # Create a RegEx object that holds the info returned from the search method
60  # using ResponseString. If this matches the pattern then parse each group.
61
62  if (MatchObject := ResponsePattern.search(ResponseString)):
63      Output = MatchObject.group(1)
64      Input = MatchObject.group(2)
65      Mode = MatchObject.group(3)
66
67      print('Input: {0}, Output: {1}, Type: {2}'.format(Input, Output, Mode))
68
```

Data Classes

Data classes are designed to make defining classes intended to store and manipulate data easier to read. Data classes automatically include an __init__ function, __repr__ function, and equality comparison functions. In most cases Extron will continue to use traditional classes for extronlib objects and in our examples/modules. Below is an example of creating and using a data class to store audio levels for a room.

```
10  from dataclasses import dataclass
11  @dataclass
12  class RoomAudioLevels:
13      '''Class for referencing default/current levels of audio in a room'''
14      roomName: str
15      HHMic1Default: float = -10
16      HHMic2Default: float = -10
17      ProgramDefault: float = -4
18      HHMic1CurLevel: float = -100
19      HHMic2CurLevel: float = -100
20      ProgramCurLevel: float = -100
21
22      def SetToDefaults(self) -> str:
23          '''Set all levels to default values'''
24          self.HHMic1CurLevel = self.HHMic1Default
25          self.HHMic2CurLevel = self.HHMic2Default
26          self.ProgramDefault = self.ProgramDefault
27          return '{} levels set to default'.format(self.roomName)
28
29  DivRoomA = RoomAudioLevels('Room 121A')
30  DivRoomB = RoomAudioLevels('Room 121B')
31
32  print(DivRoomA.SetToDefaults())
33  print(DivRoomB.SetToDefaults())
```

Changes to Regular Expressions

There is a minor change that could cause failures in some systems. Python now raises an exception when it encounters an unknown escape sequence consisting of '\' and an ASCII letter. For instance, where "\s+" was acceptable in Python 3.5, it must now be passed in as: r'\s+' or '\\s+'. In 3.5, this would have been ignored but it may be encountered in user code.

re.sub() and Pattern.sub() also enforce this change, but the unique purpose of this method means that it is not pre-compiled; therefore, they are now requiring strict escaping. Strings for pattern and repl must be properly escaped.

There is example code posted on the Extron forum, pstrptime, that may have been used in existing projects along with several Extron Device Modules that are affected. The following Extron Device Modules have been updated and are available for download. Previous instances of these modules should be replaced if the firmware of the processor is upgraded and includes the Python 3.11 upgrade.

Affected modules:

Manufacturer	Model	Module
Cisco	SX10 CE9.13.X, SX20 CE9.13.X, SX80 CE9.13.X	csco_vtc_SX_Series_CE913_v1_0_1_2
Cisco	SX10 CE9.15.X, SX20 CE9.15.X, SX80 CE9.15.X	csco_vtc_SX_Series_CE915_v1_0_0_0
Cisco	Webex Room Kit CE9.13.X, Webex Room Kit Plus CE9.13.X, Webex Room Kit Pro CE9.13.X	csco_vtc_Webex_Room_Kit_Series_CE913_v1_0_6_0
Cisco	Webex Room Kit CE9.14.X, Webex Room Kit Plus CE9.14.X, Webex Room Kit Pro CE9.14.X	csco_vtc_Webex_Room_Kit_Series_CE914_v1_3_1_0
Cisco	Webex Room Kit CE9.15.X, Webex Room Kit Plus CE9.15.X, Webex Room Kit Pro CE9.15.X	csco_vtc_Webex_Room_Kit_Series_CE915_v1_0_1_0
Cisco	Webex Room Kit Mini RoomOS 10.19.X, Webex Room Kit RoomOS 10.19.X, Webex Room Kit Plus RoomOS 10.19.X, Webex Room Kit Pro RoomOS 10.19.X	csco_vtc_Webex_Room_Kit_Series_RoomOS_10_19_v1_0_1_1
Cisco	Webex Room Kit Mini RoomOS 11.X.X, Webex Room Kit RoomOS 11.X.X, Webex Room Kit Plus RoomOS 11.X.X, Webex Room Kit Pro RoomOS 11.X.X, Webex Room Kit EQ RoomOS 11.X.X	csco_vtc_Webex_Room_Kit_Series_RoomOS_11_v1_1_0_0
Echo360	Capture Appliance, SafeCapture HD, Echo360 PRO, Echo360 POD	e360_other_CaptureAppliance_Echo360POD_Echo360PRO_ SafeCaptureHD_v1_1_10_0
Poly	G7500	poly_vtc_G7500_v1_5_0_0
Zoom	Zoom Room	zoo_vtc_Zoom_Room_V1_3_4_0

Removal of array.tostring()

The Python standard library method, array.tostring() was originally deprecated and replaced in Python 3.2. In Python 3.9, the method array.tostring() was removed from the Python standard library. When the library was deprecated, it was replaced with array.tobytes().

There are a few modules and drivers that use method array.tostring(). We are working to update these modules and drivers. If you encounter an error, you can replace array.tostring() in the module with array.tobytes() with no additional changes necessary.

Affected modules

Manufacturer	Model	Module
Bose	VB1, VB-S	bose_camera_VB1_VBS_v1_8_0_0
Bosch	Dicentis DCNM-LSYS	bsch_cs_Dicentis_DCNM_LSYS_v1_2_7_1
LEA Professional	Connect 704D, Connect 354D, Connect 164D, Connect 168D, Connect 352D, Connect 702D, Connect 84D, Connect 88D	leap_dsp_Connect_xxD_Series_v1_0_1_0
Lumens	CamConnect Pro (AI-Box1)	lumn_other_CamConnect_Pro_AI_Box1_v1_2_0_0
Tencent	Tencent Room	tenc_vtc_Tencent_Room_v1_0_1_0

Affected drivers:

Manufacturer	Model	Module
Bose	VB1, VB-S	bose_19_5035_v1_8_0
Bosch	Dicentis DCNM-LSYS	bsch_44_4230_v1_2_7
Kalyzee	Kast Revolt	klyz_19_4933_v1_1_0
LEA Professional	Connect 704D, Connect 354D, Connect 164D, Connect 168D, Connect 352D, Connect 702D, Connect 84D, Connect 88D	leap_25_4731_v1_1_0
Lumens	CamConnect Pro (AI-Box1)	lumn_31_6541_v1_2_0
Nureva	HDL300 Sound Tracking	nrva_31_5975_v1_0_1
Nureva	HDL410 Sound Tracking	nrva_31_6078_v1_0_2
Nureva	HDL310, HDL410	nrva_31_6382_v1_0_0
ROE Visual	Helios	roev_29_4810_v1_0_1
Seervision	Suite	srvs_31_6317_v1_2_0
Tencent	Tencent Room	tenc_12_6008_v1_0_1

12-2024
68-3858-01
REV. A1
Letter - English - NP