PyTine News

(Oct 13, 2020: remember to vote!)

PyTine Release 1.1.5

Client-side: Synchronous API (from last November …)

- Avoid the callback issues by using synchronous calls ?
  - **PyTine.get()** read a value without input
  - **PyTine.call()** read (and/or write) a value with/without input
  - unless the option **SYNC** is specified, the call will start an *asynchronous listener* from the buffered service layer
    - 'reflected memory'
  - make your call and update your GUI …
- **BUT**:
  - built in latency **(unless the server is scheduling the property)**
  - don't know when the data are fresh (when is a good time to call PyTine.get() ?
- Potential issue with a listener :
  - *when does it stop listening?*
    - **deadband** of 5 minutes (without a get()) will stop the listener.
      - (larger payloads have a smaller deadband)

PyTine Release 1.1.5

Client-side: Synchronous API (from last November …)

- What if a **GUI** client changes an address (*e.g. a device in a combo box*) and *knows* that it won't call the old address again?
  - waiting for a large payload to disappear via the **deadband** can lead to unnecessary network and cpu load.
    - e.g. video frames or large waveforms
- New call : **PyTine.stop_get()**
  - with same parameters as the original get() will stop the listener.

PyTine Release 1.1.5

- What's new ?
  - **Event Archive API** is now available.
  - Many Server-side features …
    - *death handlers*
    - register *your own error code*
    - assign *allowed users*
    - assign *allowed addresses*
    - *error definitions*
    - *added fields* in data dictionary in a property handler
    - *add history records*

PyTine Release 1.1.5

Event API !

PyTine.triggerEvent

Use this call if you need to trigger an event from python.

Simply typing 'PyTine.triggerEvent()' at the command prompt will generate the 'usage' exception message shown below:

```
SyntaxError: PyTine.triggerEvent(context='str',event='str',
       [comment='str',triggerLevel=val,triggerTime=val,rangeStart=val,rangeStop=val,rangeMax=val,options=val])
```

The parameter description is given in more detail below:

Parameters

context is the TINE context to which the event server receiving the trigger belongs.

event is the event trigger name. The event trigger name defines the event. A call to **triggerEvent** signals the the event server to run through the data acquistion instructions related to the event identified by this trigger name. Ergo, this must be a known event, or this call will result in an *invalid_index* return code.

comment is an optional comment to be assigned to the event at the time of the trigger. This comment will be designated as 'static' and under most circumstances will not disappear if a user further annotates the event comment for this particular event.

triggerLevel is an optional trigger level (default = 1), which indicates which step in the event data acquistion should be taken as the initial step. A value of '-1' indicates that the event server should obtain a global event number for this particular event from the *SITE* event server.

triggerTime can be used to optionally specify the UTC event id to be assigned to this event. Generally this will be the UTC value for *now* at the time of trigger, or a global value from the *SITE* event server. Under some circumstances a server might wish to e.g. collect data from hardware and post-assign a UTC time id to the event after the data have been taken.

rangeStart gives the input as a system stamp specification. A negative value or '0' signals the system to apply treat the rangeStart value as an offset to the provided 'triggerTime' as to use the input as a time stamp specification.

rangeStop provides the end of a system-stamp (or time-stamp) range to use when obtaining data froma a server's local history. If rangeStop is less that rangeStart, then the value given is used as an increment to apply to the given rangeStart (with a maximum cutoff at 1000). In other words, if rangeStop < rangeStart then rangeStop = rangeStart + rangeStop.

rangeMax provides a maximum number of sequence entries to obtain over the range provided. A negative value will negate the range specifcations entirely. A value of '0' will signal the system to use a default maximum (typically = 100).

options provides additional 'flags' to define trigger criteria. Currently, the only available option is EVNT_TRIGGER_USE_USERSTAMP which signals the event server to focus on the 'user' data stamp (and not the system stamp) in the range specifications.

As an example:

```
>>> import PyTine as pt
>>>
>>> pt.triggerEvent("TEST","sine_trigger_copy", "test triggered from pytine")
>>>
```

PyTine Release 1.1.5

Event API !

PyTine.getEventList

Use this call to obtain a list of stored events for the trigger specified.

Simply typing 'PyTine.getEventList()' at the command prompt will generate the 'usage' exception message shown below:

```
SyntaxError: PyTine.getEventList(context='str',event='str',[startTime='str',stopTime='str',limit=val])
```

The parameter description is given in more detail below:

Parameters

context is the TINE context to which the event server receiving the trigger belongs.

event is the event trigger name for which the event list is desired.

startTime optionally provides the beginning of the time range for which the event list is desired. If omitted, *now* minus *one month* will be used.

stopTime optionally provides the end of the time range for which the event list is desired. If omitted, *now* will be used.

limit optionally provides the maximum number of events to retrieve. The returned list will be in descending order. If omitted, the *limit* will be '1' (i.e. the most recent stored event).

As an example:

```
>>> import PyTine as pt
>>> pt.getEventList("TEST","sine_trigger_copy")
>>>
{'event': 'sine_trigger_copy', 'events': [{'id': 1592490377, 'time': '18.06.20 16:26:17.000 CDT'}], 'context': 'TEST'}
>>>
>>> pt.getEventList("TEST","sine_trigger_copy","10.12.2019 19:00:00", "NOW", limit=10)
>>>
{'event': 'sine_trigger_copy', 'events': [
{'time': '18.06.20 16:26:17.000 CDT', 'id': 1592490377}, {'time': '18.06.20 15:34:25.000 CDT', 'id': 1592487265},
{'time': '17.06.20 15:58:50.000 CDT', 'id': 1592402330}, {'time': '09.06.20 22:14:52.000 CDT', 'id': 1591733692},
{'time': '09.06.20 22:14:49.000 CDT', 'id': 1591733689}, {'time': '09.06.20 22:14:46.000 CDT', 'id': 1591733686},
{'time': '09.06.20 22:14:43.000 CDT', 'id': 1591733683}, {'time': '09.06.20 22:14:40.000 CDT', 'id': 1591733680},
{'time': '09.06.20 22:14:37.000 CDT', 'id': 1591733677}, {'time': '09.06.20 22:14:35.000 CDT', 'id': 1591733675}],
'context': 'TEST'}
>>>
```

PyTine Release 1.1.5

Event API !

PyTine.getEventArchiveComment

Each stored event can (and usually does) provide a comment (an annotation) describing the particular event. In most cases the initial comment is provided with the event trigger at the time of the trigger (the *static* comment). Experts who later examine the contents of the stored event can then further annotate the event in question (the *user* comment), as well as specifiy that the event should not expired (be removed from the system after some length of time), i.e. the *status*).

Simply typing 'PyTine.getEventArchiveComment()' at the command prompt will generate the 'usage' exception message shown below:

```
SyntaxError: PyTine.getEventArchiveComment(context='str',event='str',eventId=val)
```

The parameter description is given in more detail below:

Parameters

context is the TINE context to which the event server receiving the trigger belongs.

event is the event trigger name for which the event annotation is desired.

eventId is the UTC event id specifying the particular event whose data are desired. This should provide the unique UTC event Id assigned to the event when it was triggered and can be passed as a UTC integer value or a human readable string representation which corresponds to the UTC event Id.

As an example:

```
>>> import PyTine as pt
>>>
>>> pt.getEventArchiveComment(context='PETRA',event='mhf_sl0cav_trc',eventId=1563281073)
{'eventId': 1563281073, 'comment': {'static': 'PE_SL_Kly1/Treiber/Vorlaufleistung_Max/ hat getriggert', 'user': '_Thales TH-2178, SN 178004, DESY-Nr 304,
       zeigt InstabilitÃ¤t bei etwa 50 W Treiberleistung', 'status': 'SAVE'}, 'context': 'PETRA', 'event': 'mhf_sl0cav_trc'}
>>>
>>> pt.getEventArchiveComment(context='PETRA',event='mhf_sl0cav_trc',eventId='16.07.2019 14:44:33')
{'eventId': 1563281073, 'comment': {'static': 'PE_SL_Kly1/Treiber/Vorlaufleistung_Max/ hat getriggert', 'user': '_Thales TH-2178, SN 178004, DESY-Nr 304,
       zeigt InstabilitÃ¤t bei etwa 50 W Treiberleistung', 'status': 'SAVE'}, 'context': 'PETRA', 'event': 'mhf_sl0cav_trc'}
>>>
```

We see that the returned comment is actually a dictionary with *keys* **status** (the current event status - primary either 'SAVE', 'NONE' or 'DELETE'), **static** (the static event trigger comment), and **user** (the expert analysis annotation).

PyTine Release 1.1.5

Event API !

PyTine.getArchivedEventData

Use this call to actually retrieve data stored with the event.

Simply typing 'PyTine.getArchivedEventData()' at the command prompt will generate the 'usage' exception message shown below:

```
SyntaxError:
       PyTine.getArchivedEventData(context='str',event='str',eventId=val[,channel='str',eventServer='str',eventDevice='str',eventProperty='str',eventContex
       t='str',format='str',size=val])
```

The parameter description is given in more detail below:

Parameters

context	is the TINE context to which the event server receiving the trigger belongs.
event	is the event trigger name for which the event data is desired.
eventId	is the UTC event id specifying the particular event whose data are desired. This should provide the unique UTC event Id assigned to the event when it was triggered and can be passed as a UTC integer value or a human readable string representation which corresponds to the UTC event Id.
channel	optionally provides a full stored address entry string in the form /<context>/<server>/<device>[<property>]. if this parameter is not provided then the **eventServer**, **eventDevice**, and **eventProperty** parameters must be provided. The point being: the stored address of the item whose data are desired must be provided.
eventServer	optionally provides the stored server name whose data are desired (if **channel** is not used).
eventDevice	optionally provides the stored device name whose data are desired (if **channel** is not used).
eventProperty	optionally provides the stored property name whose data are desired (if **channel** is not used).
eventContext	optionally provides the stored context name whose data are desired (if **channel** is not used). If **eventContext** (and **channel**) are not provided then the value of **context** will be assumed.
format	optionally provides the desired format specification for the returned data.
size	optionally provides the desired (maximum) array size for the returned data.

As an example:

```
>>> import PyTine as pt
>>>
>>> pt.getArchivedEventData(context='TEST',event='sine_trigger_copy',eventId=1591733692,channel='/TEST/SineServer1/SineGen0[Amplitude]')
{'timestamp': 1591733692.693753, 'usrstamp': 0, 'channel': '/TEST/SineServer1/SineGen0[Amplitude]', 'status': 0, 'event': 'sine_trigger_copy', 'sysstamp':
       21134185, 'eventId': 1591733692, 'timestring': '09.06.20 22:14:52.693 CDT', 'context': 'TEST', 'data': [602.0, 233.0, 10.0, 399.0, 105.0, 322.0,
       234.0, 222.0, 281.7380065917969, 1.0]}
>>>
```

In the above, the **channel** parameter was used to specify the desired stored address and the event ID was passed directely as a UTC integer ID.

PyTine Release 1.1.5

- New server-side methods:
  - PyTine.register_death_handler()
    - *under the hood:*
      - SetSufferInSilence(TRUE)
      - SetDieAnotherDay(TRUE)

```
import PyTine as pt
import sys

msg = 'none'
def dhndlr(m):
  global msg
  print( m )
  msg = m

pt.register_death_handler( dhndlr )
pt.register_fec( name='DOOMEDFEC', port=1 )
res = pt.register_server(name='PyDoomedServer')

if (res != 0) : raise Exception( msg )
```

And it's now your responsibility to cleanup and exit !

PyTine Release 1.1.5

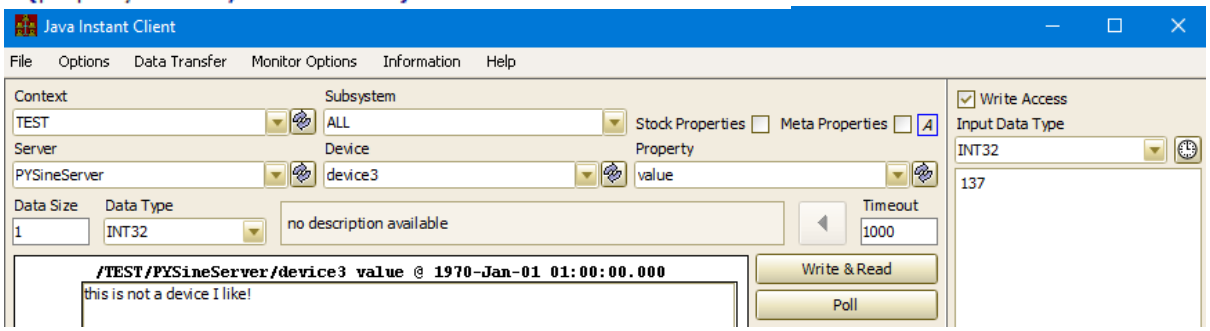
- New server-side methods:
  - PyTine.add_history_record()
    - direct API alternative to history.csv or the HIST access option in property registration.
  - TINE status codes are available :

```
import PyTine as pt
pt.register_server(name='PyTestServer')
pt.register_device(name='device0',number=0)
pt.register_device(name='device1',number=1)

pt.register_property(name='value',size=1,format='int32',mode='read|write')

pt.pushdata(property='value',device='device0',data=[42])
pt.pushdata(property='value',device='device1',data=[19])

running = True
def hndlr(inpt):
  global running
  if (not running) : return pt.command_not_accepted;
  pt.pushdata(property=inpt['property'],device=inpt['device'],data=inpt['data'])

pt.attach_handler(property='value',handler=hndlr)
```

PyTine Release 1.1.5

- New server-side methods: PyTine.register_error_code()

```
import PyTine as pt
pt.register_server(name='PyTestServer')
pt.register_device(name='device0',number=0)
pt.register_device(name='device1',number=1)
pt.register_device(name='device3',number=2)

pt.register_property(name='value',size=1,format='int32',mode='read|write')

pt.pushdata(property='value',device='device0',data=[42])
pt.pushdata(property='value',device='device1',data=[19])

not_a_device_I_like = 512
pt.register_error_code(not_a_device_I_like,"this is not a device I like!")

running = True
def hndlr(inpt):
  global running
  if (not running) : return pt.command_not_accepted;
  if (inpt['devicenumber'] == 2) : return not_a_device_I_like;
  pt.pushdata(property=inpt['property'],device=inpt['device'],data=inpt['data'])

pt.attach_handler(property='value',handler=hndlr)
```



PyTine Release 1.1.5

- New server-side methods: PyTine.errorlist()
  - PyTine.errorlist() => returns all error codes :

```
>>>
>>> pt.errorlist()
[{'text': 'success', 'code': 0}, {'text': 'illegal_line_number', 'code': 1}, {'text': 'illegal_format_specification', 'c
ode': 2}, {'text': 'illegal_arithmetic_expression', 'code': 3}, {'text': 'ambiguous_request', 'code': 4}, {'text': 'ille
gal_delimiter', 'code': 5}, {'text': 'attempt_to_divide_by_zero', 'code': 6}, {'text': 'working_area_full', 'code': 7},
{'text': 'nonexistent_name', 'code': 8}, {'text': 'transport_medium_is_invalid', 'code': 9}, {'text': 'data_size_mismatc
h', 'code': 10}, {'text': 'no_data_found_in_range', 'code': 11}, {'text': 'not_allocated', 'code': 12}, {'text': 'nonexi
stent_line_addressed', 'code': 13}, {'text': 'illegal_data_size', 'code': 14}, {'text': 'i/o_error', 'code': 15}, {'text
': 'illegal_context', 'code': 16}, {'text': 'runtime_error', 'code': 17}, {'text': 'system_error', 'code': 18}, {'text':
 'hardware_operation_in_progress', 'code': 19}, {'text': 'parameter_error', 'code': 20}, {'text': 'file_error', 'code':
21}, {'text': 'resort_to_stream_transport', 'code': 22}, {'text': 'array_dimension_error', 'code': 23}, {'text': 'square
```

  - PyTine.errorlist(code=value) (e.g.) :

```
>>>
>>>
>>> pt.errorlist(45)
{'text': 'link_timeout', 'code': 45}
>>>
>>>
```

PyTine Release 1.1.5

- New server-side methods: ACL methods
  - **PyTine.allowed_users()**
  - **PyTine.allowed_addresses()**

```
import PyTine as pt

pt.allowed_users(add=['fred','barney','wilma','betty'])

pt.allowed_users(remove='rocky')
```

in lieu of or in addition to users.csv and ipnets.csv

```
>>>
>>> pt.allowed_users()
['joe', 'fred', 'barney', 'wilma', 'duval']
>>> pt.allowed_addresses()
['131.169.9.0/24', '131.169.128.0/24', '131.169.116.0/24']
>>>
```

call without arguments just returns the current lists …

PyTine Release 1.1.5

- New server-side feature:
  - Property handler data dictionary supplies caller_address and caller_host:

```
running = True
def hndlr(inpt):
  global running
  if (not running) : return pt.command_not_accepted;
  if (inpt['devicenumber'] == 2) : return not_a_device_I_like;
  pt.pushdata(property=inpt['property'],device=inpt['device'],data=inpt['data'])

pt.attach_handler(property='value',handler=hndlr)
```

Such a property handler leads to such output :

```
>>> {'property': 'value', 'device': 'device0', 'devicenumber': 0, 'caller': 'DUVAL', 'caller_address': '131.169.9.119:61708', 'caller_host': 'localhost', 'data': 137}
>>> {'property': 'Value', 'device': 'device0', 'devicenumber': 0, 'caller': 'DUVAL', 'caller_address': '131.169.9.38:8095', 'caller_host': 'mcslxterm01.desy.de', 'data': 55}
>>>
```