MAX PLANCK
SEMICONDUCTOR
LABORATORY

EDET HOUSEKEEPING

Software & Infrastructure

Max Planck Semiconductor Lab

Martin Hensel

17.05.2022

AGENDA

- **Development progress during the last year**
  - VIServer – wrangling the gritty low-level stuff
  - VIClient – presenting useful information
  - VIManager – tying it all together, with a bow on top
  - Documentation – when everything else fails
  - Next steps – the dreaded audience participation section

- **Infrastructure**
  - Current infrastructure
  - Expected infrastructure changes (soon™)

Suggestions for a better icon are welcome, but it has to be at least 20% cooler than the current one.

DEVELOPMENT PROGRESS – VISERVER (V0.4.12)

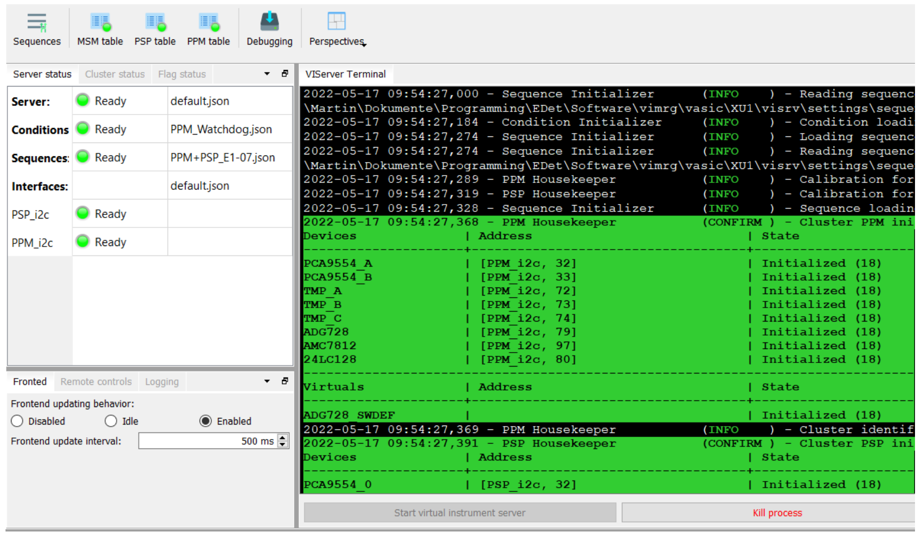
- ☑ **Structural replication of the real system with nested data and functional classes**
- ☑ **Sequencing system allowing to define complex slow-control sequences**
- ☑ **System watch-dog**
- ☑ **Real-time logging of all changes**
- ☑ **EEPROM servicing**
- ☑ **Switch definitions for all program modules on-the-fly (interfaces, clusters, sequences, conditions).**
- ☑ **Interfaces and classes for a multitude of devices (I²C, Ethernet, GPIB, file system)**
- ☑ **System independent (Win10, Ubuntu, XU1)**

- ☑ **Flexible housekeeping schedules:**
  - Define (almost) arbitrarily complex read-out or update plans with different timings.
  - Switch automatically or manually between the schedules.
- ☑ **Global value change handler and signaling:**
  Reduce update lag of and trigger recalculation for dependent terms.
- ☑ **Run multiple server instances from the same installation (as long as they have different IPs and/or ports).**
- ☑ **Bug fixing, refactoring, coding-style**
- ⟳ **Scaling calculation rework, ties into target checking as well and unifies both systems.**

DEVELOPMENT PROGRESS – VICLIENT (V0.18.4)

- ☑ Server state overview and controls
- ☑ Flexible tabular views for direct slow control
- ☑ Tabular view for sequences and their state
- ☑ View for condition checking and resetting
- ☑ View for EEPROM servicing and debugging
- ☑ Model-View architecture
- ☑ **Rework of server overview into flexible and adjustable tabbed environment**
- ☑ **Accumulation of the multitude of available views into a few easy to find categories**
- ☑ **Interface to change to server definitions**
- ☑ **Adaptation to central settings storage**
- ☑ **Adaptation to multi-server control (modularization, overview tie-ins, …)**

- ☑ **Transition from fixed PyQt5 dependency to more flexible qtpy framework:**
  - Selects either PyQt5 or PySide2 depending on availability.
  - Easy upgrade to Qt6 (hopefully).
- ⟳ **Adaptation to scaling and targeting rework (new editors for user input, changed behavior of sequences and conditions)**



Server:	● Ready	default.json
Conditions	● Ready	PPM_Watchdog.json
Sequences:	● Ready	PPM+PSP_E1-07.json
Interfaces:		default.json
PSP_i2c	● Ready	
PPM_i2c	● Ready	

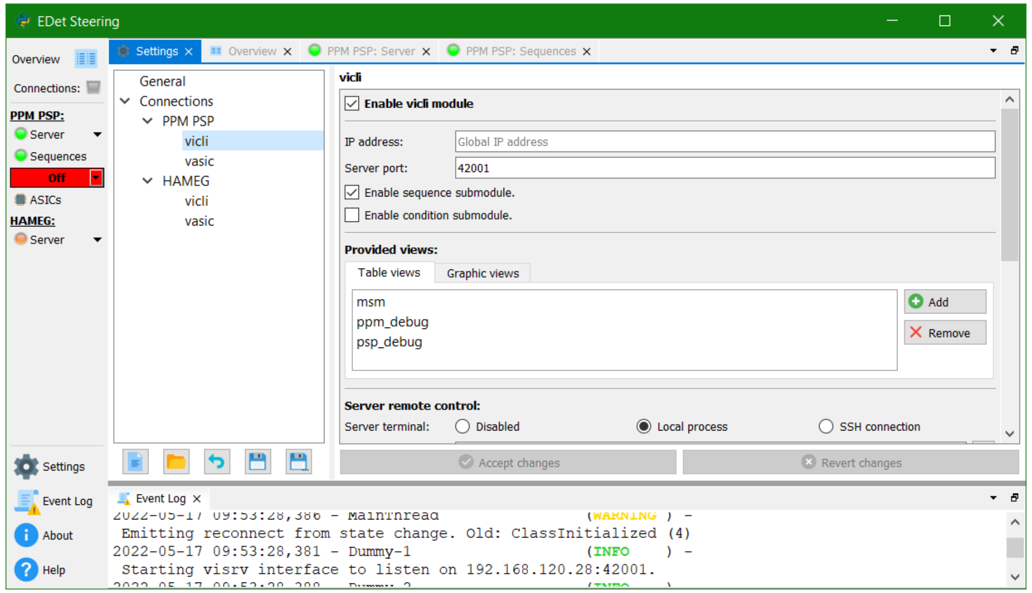
Frontend updating behavior: Disabled Idle [✓ Enabled]

Frontend update interval: 500 ms

```
2022-05-17 09:54:27,000 - Sequence Initializer      (INFO     ) - Reading sequenc
\Martin\Dokumente\Programming\EDet\Software\vimrg\vasic\XU1\visrv\settings\seque
2022-05-17 09:54:27,184 - Condition Initializer     (INFO     ) - Condition loadi
2022-05-17 09:54:27,274 - Sequence Initializer      (INFO     ) - Loading sequenc
2022-05-17 09:54:27,274 - Sequence Initializer      (INFO     ) - Reading sequenc
\Martin\Dokumente\Programming\EDet\Software\vimrg\vasic\XU1\visrv\settings\seque
2022-05-17 09:54:27,289 - PPM Housekeeper           (INFO     ) - Calibration for
2022-05-17 09:54:27,319 - PSP Housekeeper           (INFO     ) - Calibration for
2022-05-17 09:54:27,328 - Sequence Initializer      (INFO     ) - Sequence loadin
2022-05-17 09:54:27,368 - PPM Housekeeper           (CONFIRM ) - Cluster PPM ini
Devices                 | Address                              | State
------------------------+--------------------------------------+------------------
PCA9554_A               | [PPM_i2c, 32]                        | Initialized (18)
PCA9554_B               | [PPM_i2c, 33]                        | Initialized (18)
TMP_A                   | [PPM_i2c, 72]                        | Initialized (18)
TMP_B                   | [PPM_i2c, 73]                        | Initialized (18)
TMP_C                   | [PPM_i2c, 74]                        | Initialized (18)
ADG728                  | [PPM_i2c, 79]                        | Initialized (18)
AMC7812                 | [PPM_i2c, 97]                        | Initialized (18)
24LC128                 | [PPM_i2c, 80]                        | Initialized (18)
----------------------------------------------------------------------------------
Virtuals                | Address                              | State
------------------------+--------------------------------------+------------------
ADG728_SWDEF            |                                      | Initialized (18)
2022-05-17 09:54:27,369 - PPM Housekeeper           (INFO     ) - Cluster identif
2022-05-17 09:54:27,391 - PSP Housekeeper           (CONFIRM ) - Cluster PSP ini
Devices                 | Address                              | State
------------------------+--------------------------------------+------------------
PCA9554_0               | [PSP_i2c, 32]                        | Initialized (18)
```

DEVELOPMENT PROGRESS – VIMANAGER (V2.5)

- ☑ PyQt5 based container for the whole user interface, combining windows of client and ASIC/data taking → All in one tool.
- ☑ Sophisticated docking system: the user can adapt the UI to the current needs.
- ☑ Interface between slow-control, ASIC and data-taking modules.
- ☑ Provides a contained environment with all dependencies → Independent of the surrounding system.
- ☑ Tested on multiple host systems: Win10, ubuntu18, ubuntu20, Scientific Linux 7.
- ☑ **Transition to qtpy for same reasons**
- ☑ **Centrally handled settings store, providing (only) the relevant settings to all modules → Modules can have self-contained design.**
- ☑ **Container for connection-based multi-server handling: Each connected server can have its own settings, modules and views.**
- ☑ **Optional interface for external tools to control the whole GUI (too powerful at the moment, can call almost every function in the GUI, function chaining coming soon™).**



vicli

☑ Enable vicli module

IP address: Global IP address

Server port: 42001

☑ Enable sequence submodule.

☐ Enable condition submodule.

Provided views:

msm
ppm_debug
psp_debug

Server remote control:

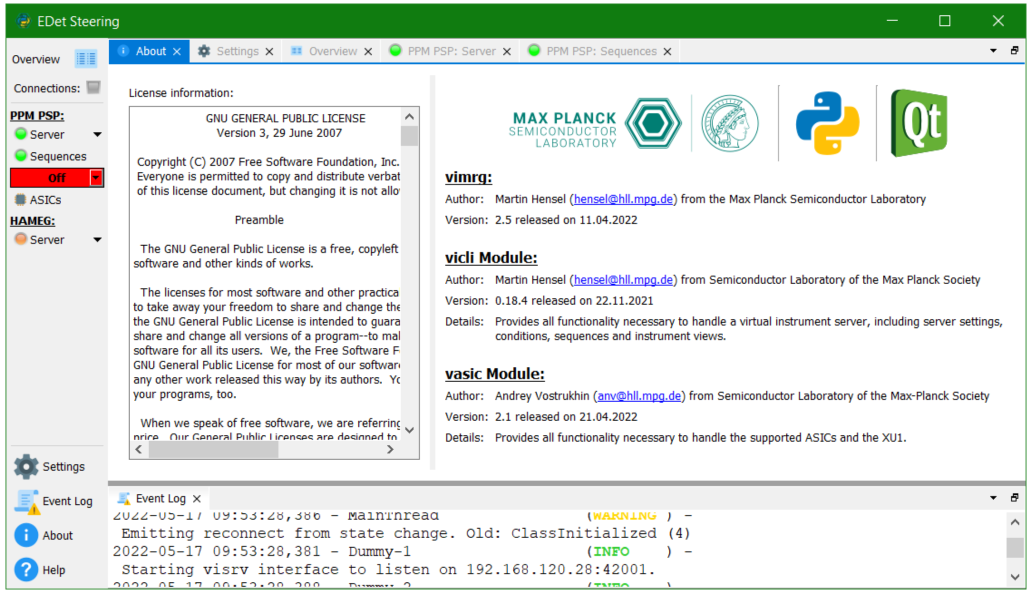
Server terminal: ○ Disabled [✓ ● Local process] ○ SSH connection

```
2022-05-17 09:53:28,386 - MainThread                    (WARNING ) -
 Emitting reconnect from state change. Old: ClassInitialized (4)
2022-05-17 09:53:28,381 - Dummy-1                       (INFO    ) -
 Starting visrv interface to listen on 192.168.120.28:42001.
```

DEVELOPMENT PROGRESS – DOCUMENTATION

- User documentation is still far away.
- Developer documentation is well on its way but patchy: It is updated and extended whenever I work on a class/function.
- Modules/classes outside of my development scope are mostly undocumented.
- Current documentation style: google
- For VSCode: Install the suggested extension “autoDocstring”, all settings are included in the workspace – helps maintaining the doc style.
- Documentation is created using the Sphinx framework that can read python inline docs as well as reStructuredText for arbitrary documentation pages.
- Documentation for now in Qt style.

- Problem: Qt decided to leave documentation on the back-burner, QtAssisstant degraded in functionality and cannot even display a styled html webpage as one would expect.
- The documentation can be provided in nice and styled html format and can even be searched with full-text search as before.



License information:

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc.
Everyone is permitted to copy and distribute verbat
of this license document, but changing it is not allo

Preamble

The GNU General Public License is a free, copyleft
software and other kinds of works.

The licenses for most software and other practica
to take away your freedom to share and change the
the GNU General Public License is intended to guara
share and change all versions of a program--to mal
software for all its users. We, the Free Software F
GNU General Public License for most of our softwar
any other work released this way by its authors. Yo
your programs, too.

When we speak of free software, we are referring

vimrg:
Author: Martin Hensel (hensel@hll.mpg.de) from the Max Planck Semiconductor Laboratory
Version: 2.5 released on 11.04.2022

vicli Module:
Author: Martin Hensel (hensel@hll.mpg.de) from Semiconductor Laboratory of the Max Planck Society
Version: 0.18.4 released on 22.11.2021
Details: Provides all functionality necessary to handle a virtual instrument server, including server settings, conditions, sequences and instrument views.

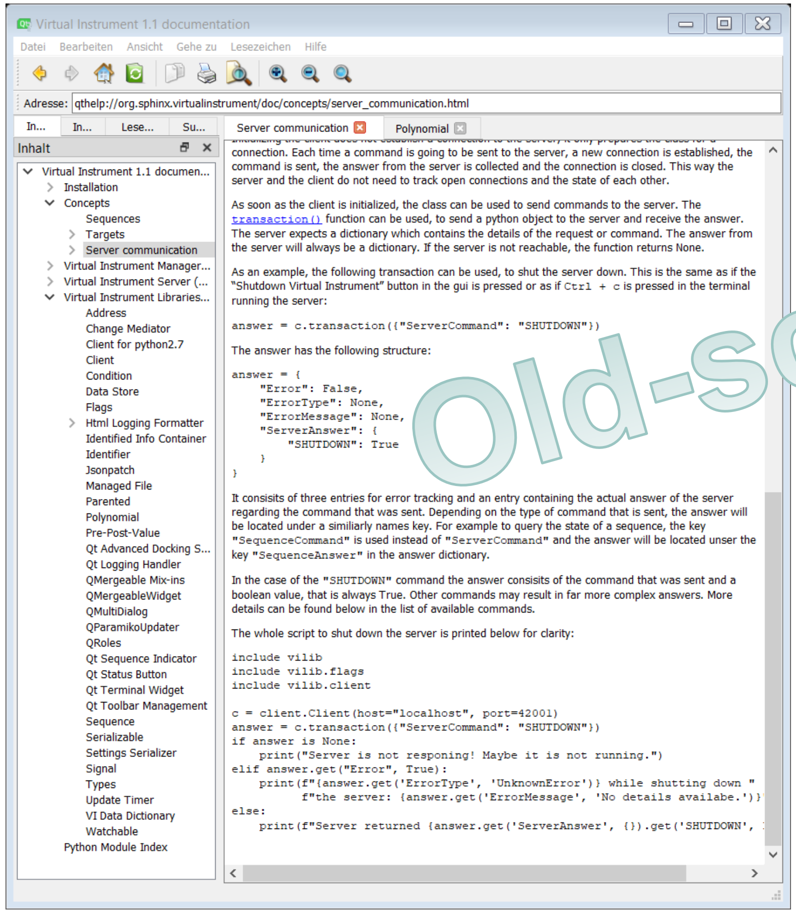
vasic Module:
Author: Andrey Vostrukhin (anv@hll.mpg.de) from Semiconductor Laboratory of the Max-Planck Society
Version: 2.1 released on 21.04.2022
Details: Provides all functionality necessary to handle the supported ASICs and the XU1.

Event Log

```
2022-05-17 09:53:28,386 - MainThread                 (WARNING ) -
 Emitting reconnect from state change. Old: ClassInitialized (4)
2022-05-17 09:53:28,381 - Dummy-1                    (INFO    ) -
 Starting visrv interface to listen on 192.168.120.28:42001.
```

DEVELOPMENT PROGRESS – DOCUMENTATION

Qt Style



connection. Each time a command is going to be sent to the server, a new connection is established, the command is sent, the answer from the server is collected and the connection is closed. This way the server and the client do not need to track open connections and the state of each other.

As soon as the client is initialized, the class can be used to send commands to the server. The `transaction()` function can be used, to send a python object to the server and receive the answer. The server expects a dictionary which contains the details of the request or command. The answer from the server will always be a dictionary. If the server is not reachable, the function returns None.

As an example, the following transaction can be used, to shut the server down. This is the same as if the "Shutdown Virtual Instrument" button in the gui is pressed or as if `Ctrl + c` is pressed in the terminal running the server:

```
answer = c.transaction({"ServerCommand": "SHUTDOWN"})
```

The answer has the following structure:

```
answer = {
    "Error": False,
    "ErrorType": None,
    "ErrorMessage": None,
    "ServerAnswer": {
        "SHUTDOWN": True
    }
}
```

It consisits of three entries for error tracking and an entry containing the actual answer of the server regarding the command that was sent. Depending on the type of command that is sent, the answer will be located under a similiarly names key. For example to query the state of a sequence, the key `"SequenceCommand"` is used instead of `"ServerCommand"` and the answer will be located unser the key `"SequenceAnswer"` in the answer dictionary.

In the case of the `"SHUTDOWN"` command the answer consisits of the command that was sent and a boolean value, that is always True. Other commands may result in far more complex answers. More details can be found below in the list of available commands.

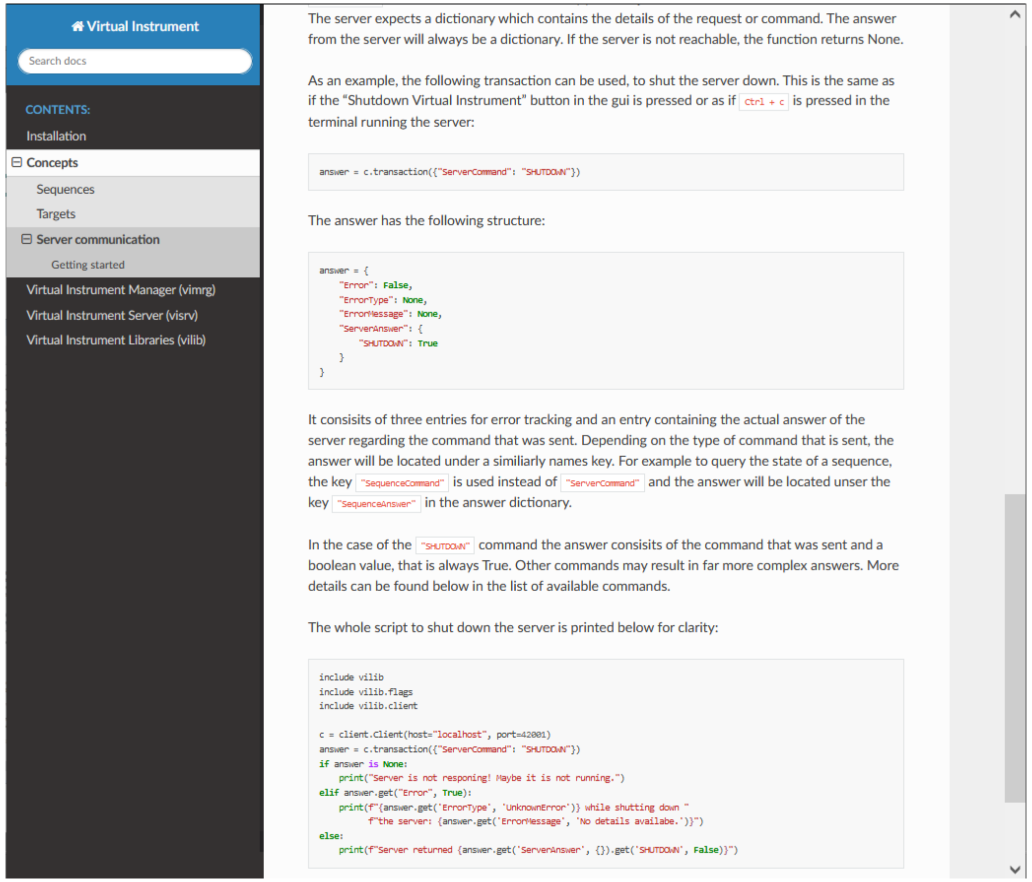
The whole script to shut down the server is printed below for clarity:

```
include vilib
include vilib.flags
include vilib.client

c = client.Client(host="localhost", port=42001)
answer = c.transaction({"ServerCommand": "SHUTDOWN"})
if answer is None:
    print("Server is not responing! Maybe it is not running.")
elif answer.get("Error", True):
    print(f"{answer.get('ErrorType', 'UnknownError')} while shutting down "
          f"the server: {answer.get('ErrorMessage', 'No details availabe.')}
else:
    print(f"Server returned {answer.get('ServerAnswer', {}).get('SHUTDOWN',
```

Old-school

Html Style



The server expects a dictionary which contains the details of the request or command. The answer from the server will always be a dictionary. If the server is not reachable, the function returns None.

As an example, the following transaction can be used, to shut the server down. This is the same as if the "Shutdown Virtual Instrument" button in the gui is pressed or as if `Ctrl + c` is pressed in the terminal running the server:

```
answer = c.transaction({"ServerCommand": "SHUTDOWN"})
```

The answer has the following structure:

```
answer = {
    "Error": False,
    "ErrorType": None,
    "ErrorMessage": None,
    "ServerAnswer": {
        "SHUTDOWN": True
    }
}
```

It consisits of three entries for error tracking and an entry containing the actual answer of the server regarding the command that was sent. Depending on the type of command that is sent, the answer will be located under a similiarly names key. For example to query the state of a sequence, the key `"SequenceCommand"` is used instead of `"ServerCommand"` and the answer will be located unser the key `"SequenceAnswer"` in the answer dictionary.

In the case of the `"SHUTDOWN"` command the answer consisits of the command that was sent and a boolean value, that is always True. Other commands may result in far more complex answers. More details can be found below in the list of available commands.

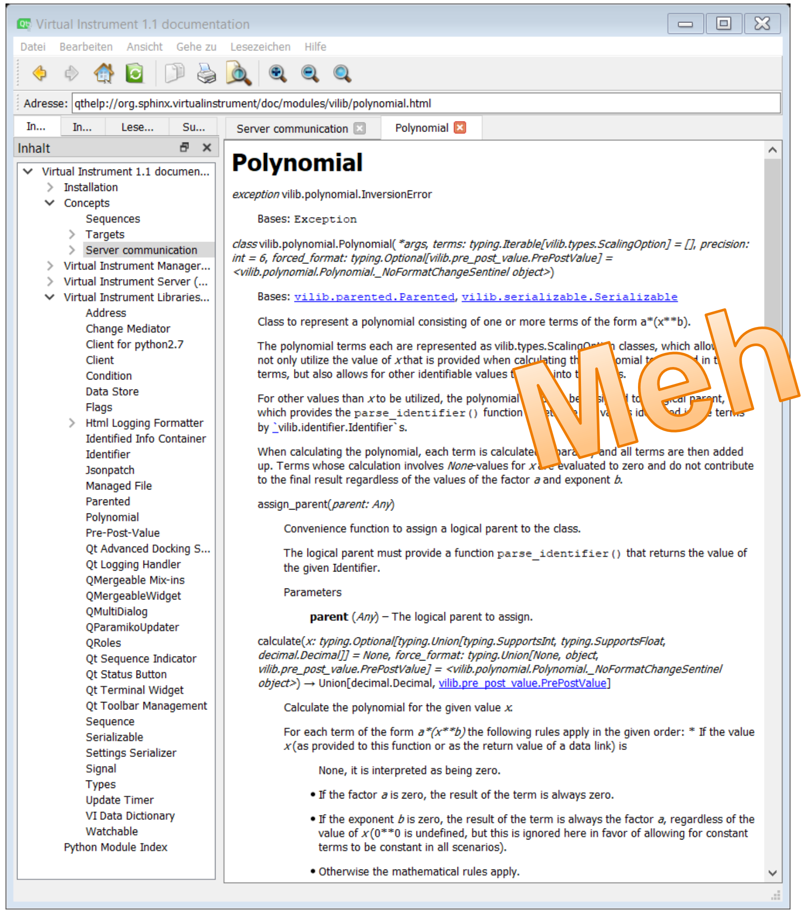
The whole script to shut down the server is printed below for clarity:

```
include vilib
include vilib.flags
include vilib.client

c = client.Client(host="localhost", port=42001)
answer = c.transaction({"ServerCommand": "SHUTDOWN"})
if answer is None:
    print("Server is not responing! Maybe it is not running.")
elif answer.get("Error", True):
    print(f"{answer.get('ErrorType', 'UnknownError')} while shutting down "
          f"the server: {answer.get('ErrorMessage', 'No details availabe.')}")
else:
    print(f"Server returned {answer.get('ServerAnswer', {}).get('SHUTDOWN', False)}")
```

DEVELOPMENT PROGRESS – DOCUMENTATION

Qt Style



Polynomial

exception vilib.polynomial.InversionError

Bases: `Exception`

class vilib.polynomial.Polynomial(**args, terms: typing.Iterable[vilib.types.ScalingOption] = [], precision: int = 6, forced_format: typing.Optional[vilib.pre_post_value.PrePostValue] = <vilib.polynomial.Polynomial._NoFormatChangeSentinel object>*)

Bases: `vilib.parented.Parented`, `vilib.serializable.Serializable`

Class to represent a polynomial consisting of one or more terms of the form a*(x**b).

The polynomial terms each are represented as vilib.types.ScalingOption classes, which allow not only utilize the value of *x* that is provided when calculating the polynomial to be used in the terms, but also allows for other identifiable values to enter into the terms.

For other values than *x* to be utilized, the polynomial needs to be assigned to a logical parent, which provides the `parse_identifier()` function to retrieve the values identified in the terms by `vilib.identifier.Identifier`s.

When calculating the polynomial, each term is calculated separately and all terms are then added up. Terms whose calculation involves *None*-values for *x* are evaluated to zero and do not contribute to the final result regardless of the values of the factor *a* and exponent *b*.

assign_parent(*parent: Any*)

Convenience function to assign a logical parent to the class.

The logical parent must provide a function `parse_identifier()` that returns the value of the given Identifier.

Parameters

parent (*Any*) – The logical parent to assign.

calculate(*x: typing.Optional[typing.Union[typing.SupportsInt, typing.SupportsFloat, decimal.Decimal]] = None, force_format: typing.Union[None, object, vilib.pre_post_value.PrePostValue] = <vilib.polynomial.Polynomial._NoFormatChangeSentinel object>*) → Union[decimal.Decimal, vilib.pre_post_value.PrePostValue]

Calculate the polynomial for the given value *x*.

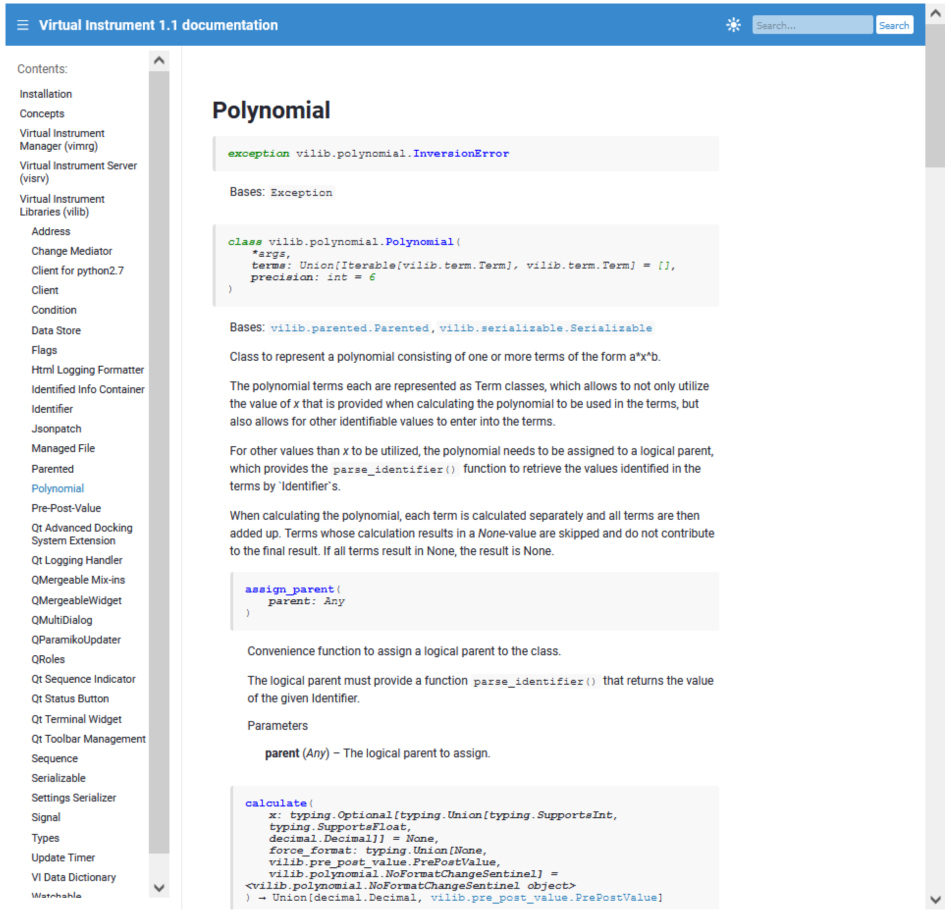
For each term of the form *a*(x**b)* the following rules apply in the given order: * If the value *x* (as provided to this function or as the return value of a data link) is

None, it is interpreted as being zero.

- If the factor *a* is zero, the result of the term is always zero.
- If the exponent *b* is zero, the result of the term is always the factor *a*, regardless of the value of *x* (0**0 is undefined, but this is ignored here in favor of allowing for constant terms to be constant in all scenarios).
- Otherwise the mathematical rules apply.

Meh / 10

Html Style



Polynomial

```
exception vilib.polynomial.InversionError
```

Bases: `Exception`

```
class vilib.polynomial.Polynomial(
    *args,
    terms: Union[Iterable[vilib.term.Term], vilib.term.Term] = [],
    precision: int = 6
)
```

Bases: `vilib.parented.Parented`, `vilib.serializable.Serializable`

Class to represent a polynomial consisting of one or more terms of the form a*x^b.

The polynomial terms each are represented as Term classes, which allows to not only utilize the value of *x* that is provided when calculating the polynomial to be used in the terms, but also allows for other identifiable values to enter into the terms.

For other values than *x* to be utilized, the polynomial needs to be assigned to a logical parent, which provides the `parse_identifier()` function to retrieve the values identified in the terms by `Identifier`s.

When calculating the polynomial, each term is calculated separately and all terms are then added up. Terms whose calculation results in a *None*-value are skipped and do not contribute to the final result. If all terms result in None, the result is None.

```
assign_parent(
    parent: Any
)
```

Convenience function to assign a logical parent to the class.

The logical parent must provide a function `parse_identifier()` that returns the value of the given Identifier.

Parameters

parent (*Any*) – The logical parent to assign.

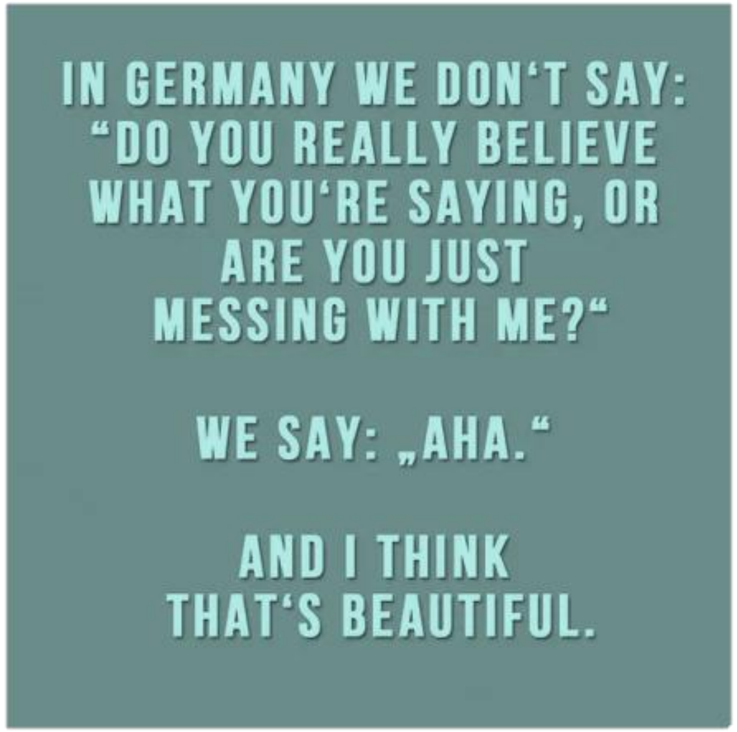
```
calculate(
    x: typing.Optional[typing.Union[typing.SupportsInt,
    typing.SupportsFloat,
    decimal.Decimal]] = None,
    force_format: typing.Union[None,
    vilib.pre_post_value.PrePostValue,
    vilib.polynomial.NoFormatChangeSentinel] =
<vilib.polynomial.NoFormatChangeSentinel object>
) → Union[decimal.Decimal, vilib.pre_post_value.PrePostValue]
```

DEVELOPMENT PROGRESS – NEXT STEPS

From the developers point of view, most critical housekeeping and slow control features seem to be implemented or are well on the way.

So now it is up to YOU to decide:

- **What do YOU want the software to do?**
- **What do YOU want it to look like?**
- **What do YOU need to be able to see at first glance and what is astonishingly irrelevant information while YOU use the software?**
- **What do YOU think needs improvement? And how do YOU think it would work better.**



IN GERMANY WE DON'T SAY: "DO YOU REALLY BELIEVE WHAT YOU'RE SAYING, OR ARE YOU JUST MESSING WITH ME?"

WE SAY: „AHA."

AND I THINK THAT'S BEAUTIFUL.

INFRASTRUCTURE – CURRENT STATE

- ☑ All(?) development progress is hosted on HLL internal git server (gitea).
- ☑ The software is distributed on several machines inside (≥8) and outside (2) of HLL.
- 😐 Outside machines need VPN access to reach internal git.
- 😐 Many developers are working on different parts of the toolchain, some are more active in committing and pushing their progress to git than others → Please be more active and adhere to `flake8` and `black` coding style!
- ☑ Issue tracking provided on the internal git as well. So long as we do not have active users outside of HLL members, that is fine.
- ☑ Installation of the software framework is automated locally in large parts, but has steps where experts are actively needed (system settings, git credentials).
- ☑ The framework comes with its own python and Qt versions inside a (Ana)conda environment → mostly independent of other software on the target machine.
- ☑ Large installers are hosted outside the git (blew repo up to 5GB, now back to 175MB) using MPG Keeper and a two-step download process to fetch the latest release.

The Gitea database holds:
18 users, 20 public keys, 31 repositories, 258 watches, 11064 actions, 186 accesses, 47 issues, 187 comments, 33 releases, 6 milestones, 49 labels, 7 teams, 3 attachments.

INFRASTRUCTURE – EXPECTED CHANGES

Source control:

- **Move internal git to externally accessible and hosted service so that external machines loose the VPN dependence.**
- **GWDG hosts a GitLab, but currently large changes are applied to allowed usage and features.**
- **MPDL is in slow negotiations with GitLab for a MPG wide instance as well.**
- **HLL might acquire its own GitLab or GitHub instance hosted by GWDG for the foreseeable future.**
- **In addition this allows contributions/issue raising from our users outside the HLL.**

System setup:

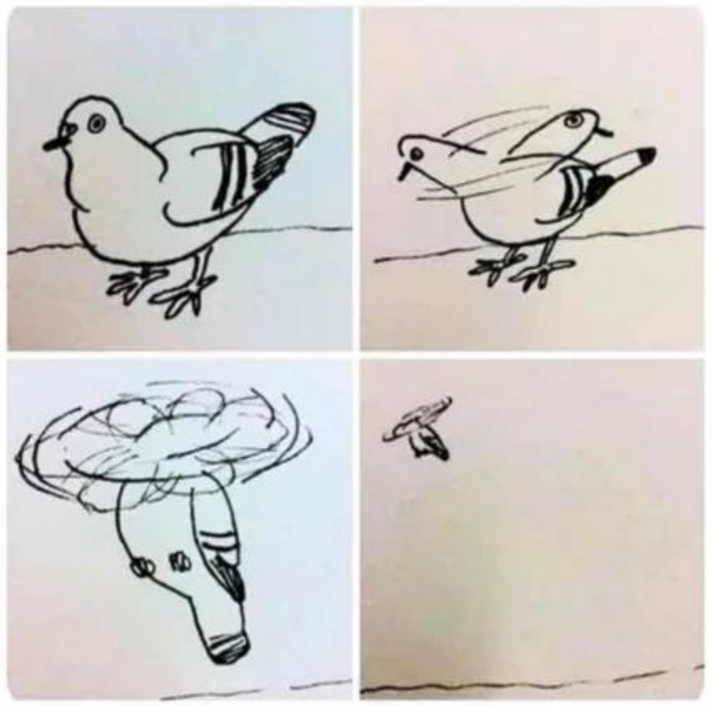
- **Develop an automated installation and upgrade process that can be triggered on remote machines.**
- **At the moment ANSIBLE is investigated for handling these tasks:**
  - Allows remote setup of the whole machine (as soon as the operating system is available and a connection to the HLL net is established) as well as all software related installation tasks.
  - Allows triggering updates on remote machines.
- **Usability outside of the HLL net?**

REQUESTS?

QUESTIONS?

BUG-REPORTS?

When your program is a complete mess, but it does its job



THANK YOU FOR YOUR ATTENTION

For questions please contact:

Max Planck Society – Semiconductor Laboratory

Martin Hensel

Otto-Hahn-Ring 6
81739 München

Tel.: +49 (0)89 839400-0
Fax: +49 (0)89 839400-11

E-Mail: hll-info@hll.mpg.de
Internet: www.hll.mpg.de