Quick Start Guide PrismaPro API

Introduction

This API Guide provides an overview on how to use the PrismaPro API along with some simple examples. For details on every feature of the API, please refer to the **API Reference PrismaPro** (downloadable from the **PV-Cloud**). Futhermore, you will be shown additional options for communicating with the PrismaPro.

Any command noted here is intended for the PrismaPro.

Please read this entire guide before executing any commands. It is recommended that one only use commands necessary for their desired control.

Note: To "set" or "run" API's is at your own risk!

API:	***Special Commands to communicate with PrismaPro***
PVMassSpec:	***User-friendly Windows software for PrismaPro***
MPP:	***PrismaPro Driver to communicate with PLC's (Siemens S7) and OPC-UA-Server***

Download links in our PV-Cloud

Here you'll find a lot of documentation and examples around the API-Communication with the PrismaPro. Also the MPP-Software and Documentation, the PVMassSpecc-Software and the PrismaPro Firmware:

API:	**https://cloud.pfeiffer-vacuum.de/index.php/s/M8Lfhztew9Y5JXH** **Password: PrismaPro**
PVMassSpec: & Firmware:	**https://cloud.pfeiffer-vacuum.de/index.php/s/HyQMI5iyW1u5TRu** **Password: PrismaPro**
MPP:	**https://cloud.pfeiffer-vacuum.de/index.php/s/ZT9laa4UpSTXphq** **Password: PrismaPro**

Quick Start Example

Install Firefox or Chrome and install JSONView. Then enter the following nine URLs into your browser (e.g. one per tab), preceded by **`http://ipAddress:port`**

Setting the emission on can take several seconds, so we do it first. **WARNING!** The sensor MUST be under vacuum when emission is turned on, or the filament will be destroyed.

```
/mmsp/generalControl/set?setEmission=on
```

Before setting up the channels, turn off scanning.

```
/mmsp/scanSetup/set?scanStop=Immediately
```

The next three commands configure channel 1 to sweep through masses 0 to 30, measure 4 points at each mass, and dwell on each point for 32 milliseconds. The total number of data points generated per scan will be 1 + (30 - 0) * 4 = 121. These three commands could be combined into a single command, but are broken up into multiple lines here for formatting:

```
/mmsp/scanSetup/channels/1/set?channelMode=Sweep
/mmsp/scanSetup/channels/1/set?startMass=0&stopMass=30
/mmsp/scanSetup/channels/1/set?dwell=32&ppamu=4&enabled=True
```

Channels must not only be enabled (done above) to take effect, but they must also be within range. In this example we use only one channel:

```
/mmsp/scanSetup/set?startChannel=1&stopChannel=1
```

Setting scanCount to -1 tells the instrument to continue scanning indefinitely:

```
/mmsp/scanSetup/set?scanCount=-1
```

Start scanning:

```
/mmsp/scanSetup/set?scanStart=1
```

Now we can repeatedly get data from the most recently completed scan. There should be 121 values in the response, and a new scan should appear after every (121 * 32) milliseconds, or about every four seconds.

```
/mmsp/measurement/scans/-1/get
```

To retrieve current data while a scan is in progress, get element 0 of the scan instead of -1, or use the /mmsp/measurement/data target.

HTTP and JSON

The PrismaPro MPP is a web server whose API is accessed over HTTP.
This makes it possible for standard web browsers to control the MPP, and the API is also designed for any TCP/IP-capable program to be able to control the instrument.

Although most programing environments provide well-tested standard modules for HTTP client communication, some developers may want to implement their own HTTP communication code on top of a TCP/IP stack. Complete details of HTTP are officially described here, and simple examples are widely available e.g. on Wikipedia.

The API is designed to be accessible from high-level scripting languages or low-level C code. It can be controlled from JavaScript running in web browsers, Python, Java, C# or Visual Basic. To simplify development of client programs in any language, data is sent as JSON, a lightweight data-interchange format that's easy for humans to read and write AND also easy for machines to parse and generate.

Using JSON over HTTP is not the most compact or efficient protocol possible. Some speed and bandwidth is sacrificed in favor of clarity, flexibility, and simplicity of client development. JSON makes it easier to develop software that is compatible across multiple revisions of the API, and easier for developers to understand and debug API requests and responses. But JSON also makes communication messages larger, and it requires more processing to read and write.

Fortunately, the additional overhead imposed by JSON compared to an optimized binary protocol is usually negligible. Most API calls involve small amounts of data where performance limits are dominated by low level I/O latency and minimum packet sizes. Performance is therefore not significantly affected by the addition of a few hundred bytes per message. One exception, however, is the generation of large amounts of scan data. For this a binary option is provided.

HTTP Response Details

MPP responses are compatible with standard HTTP clients, but they are modified to allow for simplified processing by hand coded HTTP clients if desired.

Below is a header returned from the MPP server. The fields are padded with blanks on the right so that the header is always (at least) 189 bytes and the field values always start at the same position. (We may add additional fields later, but these fields will always be at the start in this order.)

```
HTTP/1.1 200  OK                             \r\n
X-Content-Mode: full            \r\n
X-Header-Length: 225       \r\n
Content-Type: application/json               \r\n
Content-Length: 71        \r\n
Date: Fri Aug 26 11:52:27 2011 GMT\r\n

The positions of the field information:
    Response Code (200):             9
    Response Message (OK):          14
    Content Mode (full or chunked):  64
    Header Length (225):            99
    Content Type (app/json):       125
    Content Length (71):           175 if Mode is full
    Transfer Encoding (chunked):   178 if Mode is chunked
    Date (Fri, Aug 26 ..):         193
```

Most messages are not chunked. Large messages (e.g. multiple scans) may be chunked. Requests for a single scan are not chunked, regardless of size.

Chunked messages have their chunk sizes padded on the left with zeros to be 4 characters wide.

The algorithm for reading messages with minimal calls to recv():

```
1. Read 225 bytes.
2. If byte 64 is 'f' (full), read header-length (byte 99+)
                                 + content-length (byte 125+)
                                 - 225 bytes.
    If byte 64 is 'c' (chunked):
       Read header-length (byte 99+) + 8 - 225 bytes.
       If last 8 bytes are "0000\r\n\r\n", end of message.
       Otherwise, last 8 bytes are "XXXX\r\n..",
                       where XXXX is hex chunk size.
                       (.. are first 2 bytes of chunk)
       Read chunk size - 2 + 8 bytes for next chunk plus following size.
       Repeat.
```

Note that each value in scan data is 12 characters wide. If it is floating point, it will be of the format "N.NNNNNNe-NN". Integer values will be padded on the left to 12 characters.

EventBus Details

EventBus messages are designed to be compatible with internal SCXML events, and with HTTP URLs. Their canonical format is JSON, but a simple event can also be represented as a URL in a single line of text, or (in the future) as an XML structure. Events have the following structure:

Property name	**Type**	
name	dotted.string	required
data	any JSON	optional
target	URL path	required, but often implicit
origin	URL path	optional, often implicit
sendid	string	optional, rarely used

When the data structure for an event is simple enough, it can be sent as a single URL string, i.e.

```
    http://localhost:8899/targetName/sys.abort?code=27&severity=red
```

The final segment of the path (between the last "/" and the "?") is the event name. The example above is equivalent to sending the JSON string

```
    {"name":"sys.abort", "data":{"code":27,"severity":"red"}}
```

to the target `http://localhost:8899/targetName`

Targets may be composed of multiple segments separated by slashes. For example, in the URL

```
    http://localhost:8899/partA/partB/partC/foo?x=100
```

the target is `/partA/partB/partC`, and the event name is `foo`.

Event Response

Upon completion of an HTTP EventBus request, the client may receive any of the usual HTTP status codes, including 200 OK for success. The server may optionally send one or more JSON events in the body of the response. If the server returns a 200 OK with no body, or a non-event body, the client must consider this transaction a successful event delivery. In case of errors the server may return an HTTP status error (e.g. 404 Not Found), or it may return a 200 OK with a JSON error event in the body.

EventBus Errors

Any event with a **name** that begins with the string **error** (for example **error.missingParameter**) is considered an error. Error events can include additional information in their **data** property, but note that English text in **data.message** is only for use by developers, and should not be displayed to end users, because user messages are looked up based on language. HTTP servers should avoid responding with HTTP error codes for EventBus-level errors, because HTTP client libraries tend to catch these and get in the way of reading event details. HTTP error codes should be reserved for lower level issues, and EventBus error events should have status 200.

A response event must be JSON and include the **name** property, while all other properties are optional. If the response includes an **origin** property, this should be a valid target. Servers should publish events using the standard MIME content-type "application/json" when the client includes this in the accept header, but clients must also accept JSON body responses with type "text/plain".

EventBus APIs

An EventBus server API consists of a set of valid targets, the set of event names and data that each target accepts, and their associated response events. An API may be documented informally, or it may be described formally with a JSON Schema description. The PrismaPro JSON Schema API Description is available from the URL `/mmsp/get.api`

Examples

Two simple examples are shown below - a scan of a range of masses (Monitor) and a scan of a single data point (Leakcheck). The third is a recipe via cURL and startet from a batch-File (DOS-cmd)

Monitor Example

This example sets up continuous scans from mass 0 to 50. Values from each scan will begin with the timestamp and total pressure, followed by datapoints for each mass.

```
/mmsp/scanSetup/set?scanStop=Immediately
/mmsp/scanSetup/channels/1/set?channelMode=Timestamp&enabled=True
/mmsp/scanSetup/channels/2/set?channelMode=TotalPressure&enabled=True
/mmsp/scanSetup/channels/3/set?channelMode=Sweep&startMass=0&stopMass=50
/mmsp/scanSetup/channels/3/set?dwell=32&ppamu=10&enabled=True
/mmsp/scanSetup/set?startChannel=1&stopChannel=3
/mmsp/scanSetup/set?scanCount=-1
/mmsp/generalControl/set?setEmission=on
/mmsp/scanSetup/set?scanStart=1
/mmsp/measurement/scans/-1/get
```

Leakcheck Example

This example checks only for the presence of helium, so scans will only include a single data point.

```
/mmsp/scanSetup/set?scanStop=Immediately
/mmsp/scanSetup/channels/4/set?channelMode=Single&startMass=4&dwell=32&enabled=True
/mmsp/scanSetup/set?startChannel=4&stopChannel=4
/mmsp/scanSetup/set?scanCount=-1
/mmsp/generalControl/set?setEmission=on
/mmsp/scanSetup/set?scanStart=1
/mmsp/measurement/scans/-1/get
```

Set recipe (1 channel only)

```
@echo off
echo PRISMA PRO Recipe Sweep 0-100 on 192.168.1.100
echo ====================================================
timeout /t 1 >nul
curl 192.168.1.100/mmsp/scanSetup/channel/1/channelMode/set?Sweep
curl 192.168.1.100/mmsp/scanSetup/channel/1/startMass/set?0
curl 192.168.1.100/mmsp/scanSetup/channel/1/StopMass/set?100
curl 192.168.1.100/mmsp/scanSetup/channel/1/dwell/set?32
curl 192.168.1.100/mmsp/scanSetup/channel/1/ppamu/set?5
curl 192.168.1.100/mmsp/scanSetup/channel/1/enabled/set?True
curl 192.168.1.100/mmsp/scanSetup/startChannel/set?1
curl 192.168.1.100/mmsp/scanSetup/stopChannel/set?1
curl 192.168.1.100/mmsp/scanSetup/scanCount/set?-1
cmd /V:On
```

Programming examples:

If your programming language doesn't have built-in support for JSON, you need the right JSON parser.

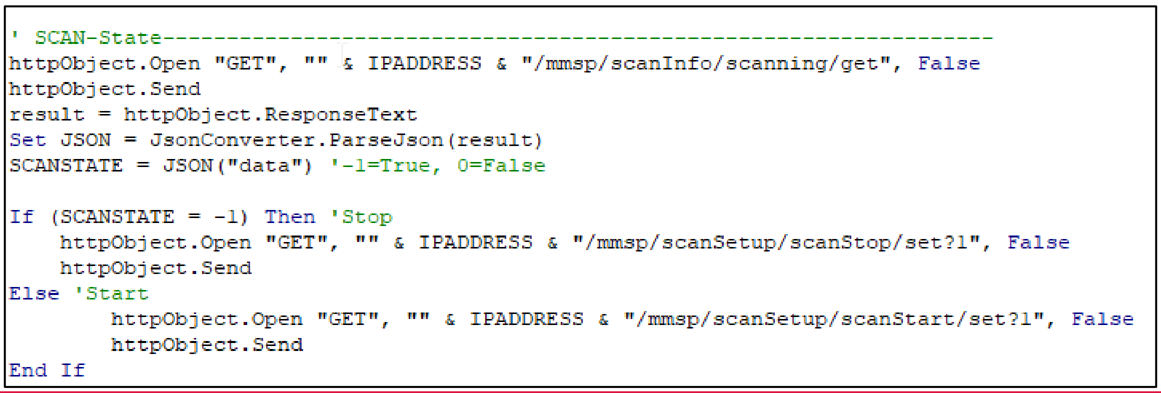
www.json.org

VBA

See the Excel file "**PrismaPro_API.xlsm**" (downloadable from the **PV-Cloud**)
TIP: [ALT]+[F11] ➔ *Open the VBA-View*

Example:

```
Private Sub cmdEmission_Click()
Dim httpObject As Object
Dim i As Integer
Dim result As String
Dim EMISSION As String
On Error GoTo fault
Set httpObject = CreateObject("MSXML2.XMLHTTP")
Dim JSON As Object
' Toggle Emission---------------------------------------------------------------
httpObject.Open "GET", "" & "192.168.1.100" & "/mmsp/generalControl/setEmission/get", False
httpObject.Send
result = httpObject.ResponseText
Set JSON = JsonConverter.ParseJson(result)  ' *****JSON-PARSER*****
EMISSION = JSON("data")
If (EMISSION = "On") Then
   httpObject.Open "GET", "" & "192.168.1.100" & "/mmsp/generalControl/setEmission/set?Off", False
   httpObject.Send
   Else
   httpObject.Open "GET", "" & "192.168.1.100" & "/mmsp/generalControl/setEmission/set?On", False
   httpObject.Send
End If
Exit Sub
'Fault????
fault:
   MsgBox ("No/Wrong Answer from PrismaPro! =>Error: " & ERROR)
End Sub
```



```
' SCAN-State---------------------------------------------------------------------------
httpObject.Open "GET", "" & IPADDRESS & "/mmsp/scanInfo/scanning/get", False
httpObject.Send
result = httpObject.ResponseText
Set JSON = JsonConverter.ParseJson(result)
SCANSTATE = JSON("data") '-1=True, 0=False

If (SCANSTATE = -1) Then 'Stop
    httpObject.Open "GET", "" & IPADDRESS & "/mmsp/scanSetup/scanStop/set?1", False
    httpObject.Send
Else 'Start
        httpObject.Open "GET", "" & IPADDRESS & "/mmsp/scanSetup/scanStart/set?1", False
        httpObject.Send
End If
```

Python

The Python programming language provides a simple way to control the PrismaPro, because it has built-in support for HTTP and JSON. An example program is provided at the **PV-Cloud** to set up scans and acquire large amounts of data into a CSV spreadsheet file.

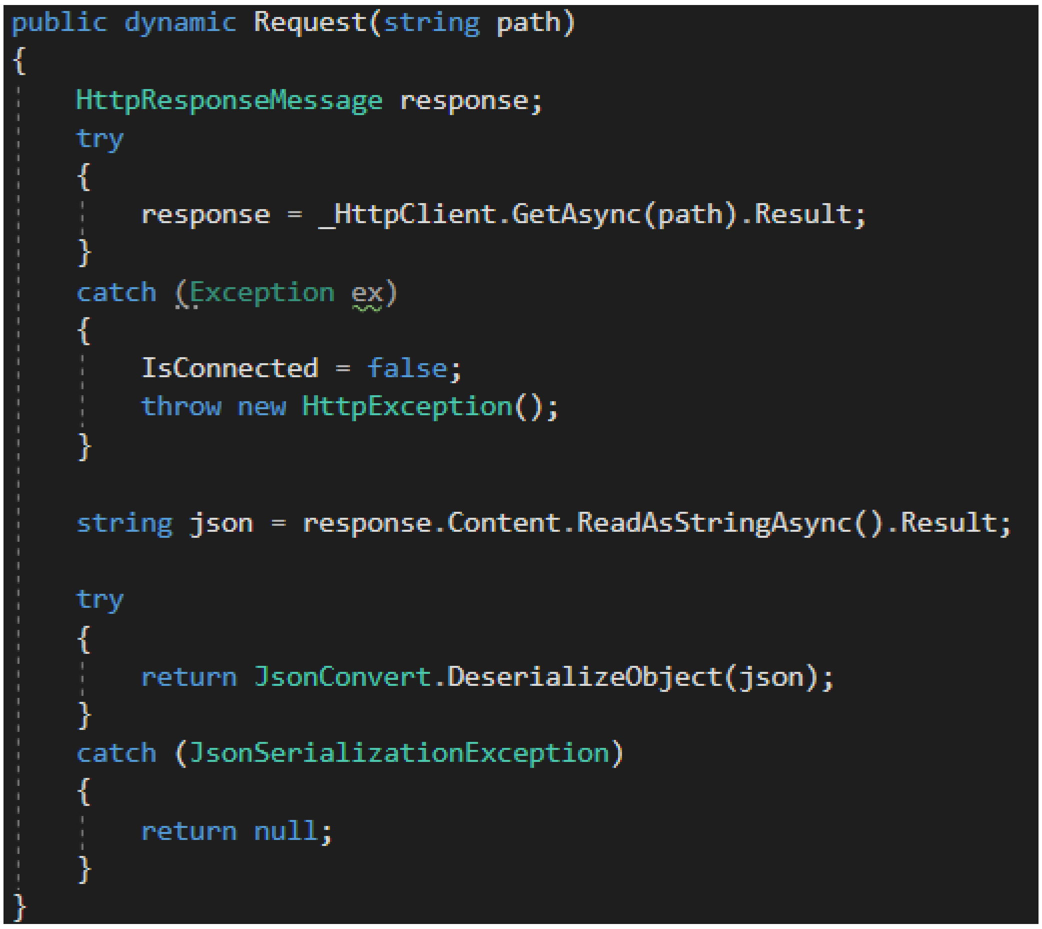
Extract from an example:

```
def start_scan(mmsp, channels):
  # Take control to the unit and make sure it's stopped
   print("Taking control...")
   mmsp.send("/mmsp/communication/control/set?force")
   mmsp.send('/mmsp/scanSetup/set?scanstop=1')
   # Setup the channels
   print("Setting up channels...")
   chan_num = 0
   for c in channels:       chan_num += 1
       dwell = DEFAULT_DWELL
       if type(c) == str:
           print("Channel #%d: %s" % (chan_num, c))
           mmsp.send("/mmsp/scanSetup/set?@channel=%d&channelMode=%s&dwell=8&enabled=True" % (chan_num, c))
       elif isinstance(c, SingleMass):
           if c.dwell is not None:
               dwell = c.dwell
           print("Channel #%d: Single Mass %f [dwell=%d]" % (chan_num, c.mass, dwell))
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&channelMode=Single' % chan_num)
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&enabled=True' % chan_num)
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&startmass=%.1f' % (chan_num, c.mass))
          mmsp.send('/mmsp/scanSetup/set?@channel=%d&dwell=%d' % (chan_num, dwell))
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&ppamu=1' % chan_num)
       elif isinstance(c, MassSweep):
           if c.dwell is not None:
               dwell = c.dwell
           print("Channel #%d: Sweep %f..%f [dwell=%d, ppamu=%d]" % (chan_num, c.start, c.stop, dwell, c.ppamu))
          mmsp.send('/mmsp/scanSetup/set?@channel=%d&channelMode=Sweep' % chan_num)
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&enabled=True' % chan_num)
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&startmass=%.1f' % (chan_num, c.start))
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&stopmass=%.1f' % (chan_num, c.stop))
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&dwell=%d' % (chan_num, dwell))
           mmsp.send('/mmsp/scanSetup/set?@channel=%d&ppamu=%d' % (chan_num, c.ppamu))
       else:
           print("Don't know how to handle", type(c), c)
   # Start the scan
   print("Starting Scan...")
   mmsp.send("/mmsp/scanSetup/set?startchannel=1&stopchannel=%d" % chan_num)
   mmsp.send("/mmsp/scanSetup/set?scancount=%d" % SCAN_COUNT)
   mmsp.send("/mmsp/scanSetup/CaptureFileName/set?%s" % CAPTURE_FILE)
   mmsp.send("/mmsp/scanSetup/set?scanInterval=%d&scanstart=1" % SCAN_INTERVAL)
         ....
```

C#

<u>Extract from an example *(MPP-Application is written in C#):*</u>

standard http components:

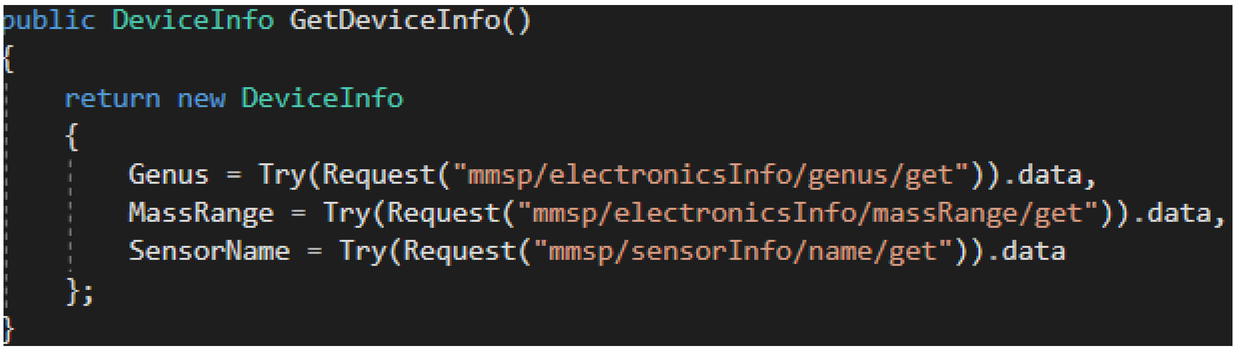


```
public dynamic Request(string path)
{
    HttpResponseMessage response;
    try
    {
        response = _HttpClient.GetAsync(path).Result;
    }
    catch (Exception ex)
    {
        IsConnected = false;
        throw new HttpException();
    }

    string json = response.Content.ReadAsStringAsync().Result;

    try
    {
        return JsonConvert.DeserializeObject(json);
    }
    catch (JsonSerializationException)
    {
        return null;
    }
}
```

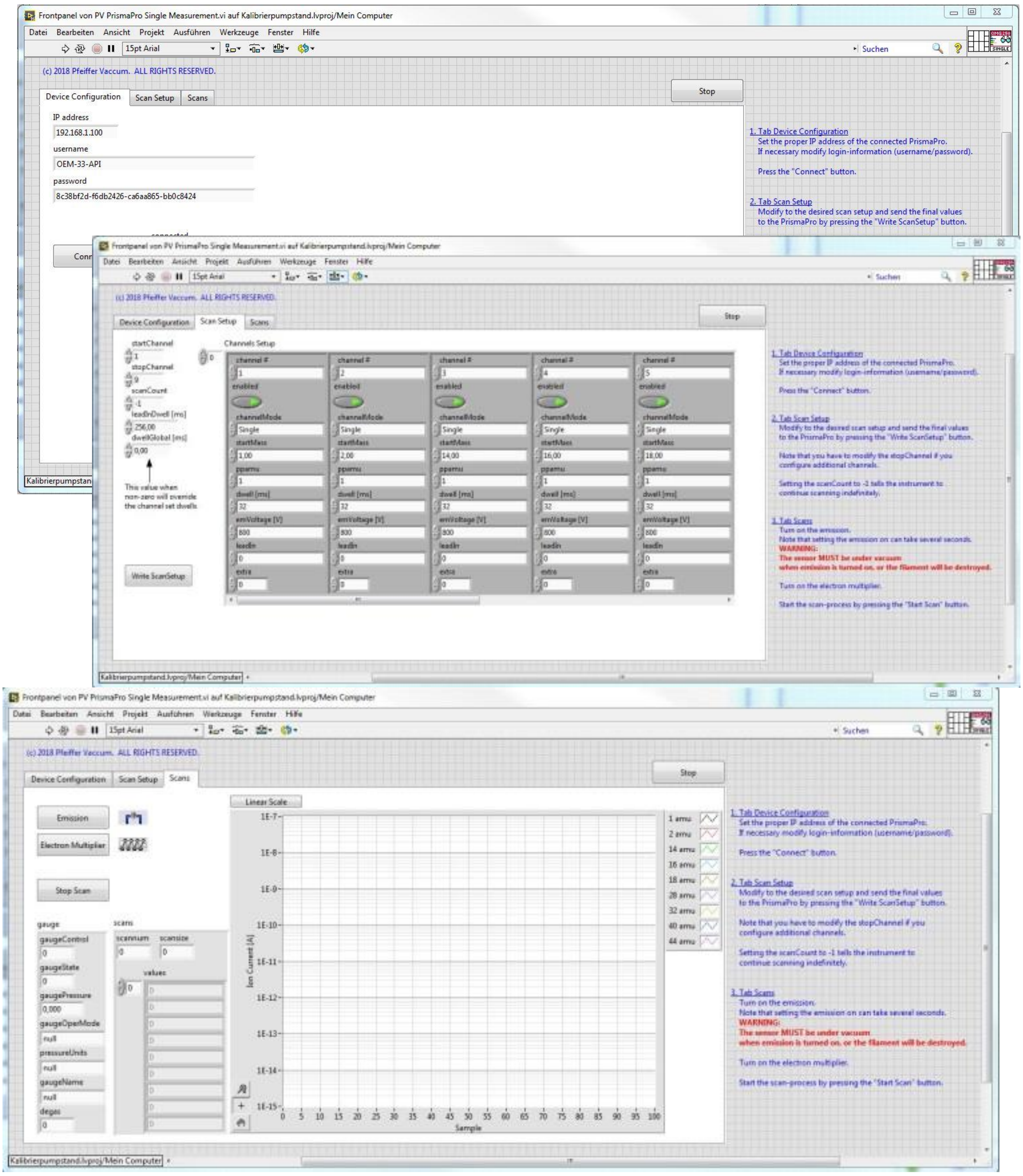
Usage the API:



```
public DeviceInfo GetDeviceInfo()
{
    return new DeviceInfo
    {
        Genus = Try(Request("mmsp/electronicsInfo/genus/get")).data,
        MassRange = Try(Request("mmsp/electronicsInfo/massRange/get")).data,
        SensorName = Try(Request("mmsp/sensorInfo/name/get")).data
    };
}
```

LabView:

A LabView-Driver is available in the PV-Cloud.
Togther with this example and the **API Reference PrismaPro** you can write your own VI's.


Frontpanel von PV PrismaPro Single Measurement.vi auf Kalibrierpumpstand.lvproj/Mein Computer
Datei Bearbeiten Ansicht Projekt Ausführen Werkzeuge Fenster Hilfe
15pt Arial
Suchen
Device Configuration
Scan Setup
Scans
Stop
Connect
Write ScanSetup
Emission
Electron Multiplier
Stop Scan
Linear Scale
Kalibrierpumpstand.lvproj/Mein Computer

(c) 2018 Pfeiffer Vaccum. ALL RIGHTS RESERVED.

IP address: 192.168.1.100
username: OEM-33-API
password: 8c38bf2d-f6db2426-ca6aa865-bb0c8424

startChannel: 1
stopChannel: 9
scanCount: -1
leadInDwell [ms]: 256,00
dwellGlobal [ms]: 0,00

This value when non-zero will override the channel set dwells

Channels Setup

channel #	1	2	3	4	5
channelMode	Single	Single	Single	Single	Single
startMass	1,00	2,00	14,00	16,00	18,00
ppamu	1	1	1	1	1
dwell [ms]	32	32	32	32	32
emVoltage [V]	800	800	800	800	800
leadIn	0	0	0	0	0
extra	0	0	0	0	0

gauge: gaugeControl 0; gaugeState 0; gaugePressure 0,000; gaugeOperMode null; pressureUnits null; gaugeName null; degas 0

scans: scannum 0; scansize 0; values 0

Ion Current [A]: 1E-7, 1E-8, 1E-9, 1E-10, 1E-11, 1E-12, 1E-13, 1E-14, 1E-15
Sample: 0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
1 amu, 2 amu, 14 amu, 16 amu, 18 amu, 28 amu, 32 amu, 40 amu, 44 amu

1. Tab Device Configuration
Set the proper IP address of the connected PrismaPro.
If necessary modify login-information (username/password).

Press the "Connect" button.

2. Tab Scan Setup
Modify to the desired scan setup and send the final values to the PrismaPro by pressing the "Write ScanSetup" button.

Note that you have to modify the stopChannel if you configure additional channels.

Setting the scanCount to -1 tells the instrument to continue scanning indefinitely.

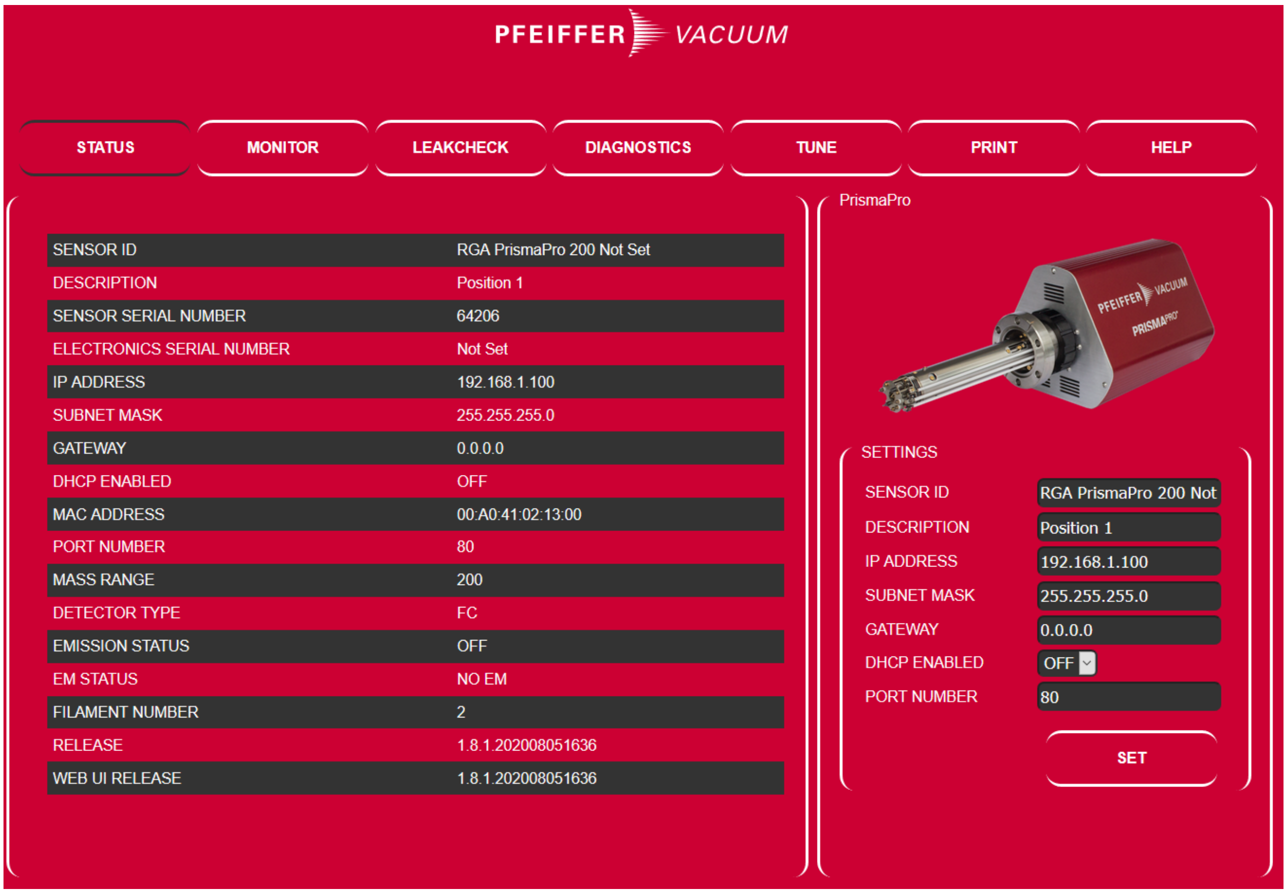
3. Tab Scans
Turn on the emission.
Note that setting the emission on can take several seconds.
WARNING:
The sensor MUST be under vacuum
when emission is turned on, or the filament will be destroyed.

Turn on the electron multiplier.

Start the scan-process by pressing the "Start Scan" button.

Further Communication options:

1. **WebUI** (*http://PrismaPro-IP-Address*)
   (**http://192.168.1.100**) (= Standard Address)

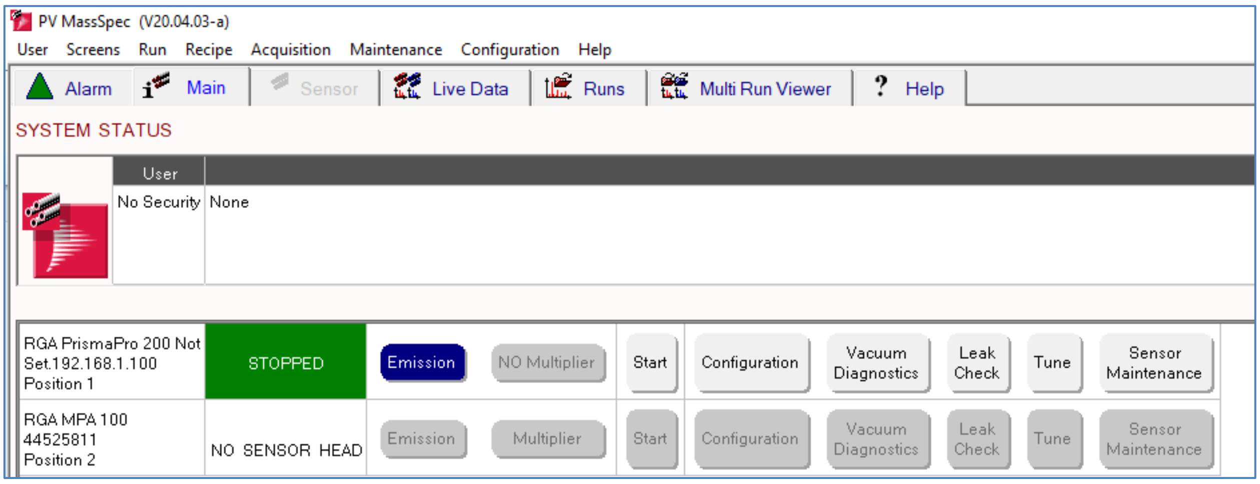


SENSOR ID	RGA PrismaPro 200 Not Set
DESCRIPTION	Position 1
SENSOR SERIAL NUMBER	64206
ELECTRONICS SERIAL NUMBER	Not Set
IP ADDRESS	192.168.1.100
SUBNET MASK	255.255.255.0
GATEWAY	0.0.0.0
DHCP ENABLED	OFF
MAC ADDRESS	00:A0:41:02:13:00
PORT NUMBER	80
MASS RANGE	200
DETECTOR TYPE	FC
EMISSION STATUS	OFF
EM STATUS	NO EM
FILAMENT NUMBER	2
RELEASE	1.8.1.202008051636
WEB UI RELEASE	1.8.1.202008051636

SETTINGS

SENSOR ID	RGA PrismaPro 200 Not
DESCRIPTION	Position 1
IP ADDRESS	192.168.1.100
SUBNET MASK	255.255.255.0
GATEWAY	0.0.0.0
DHCP ENABLED	OFF
PORT NUMBER	80

2. **PVMassSpec**



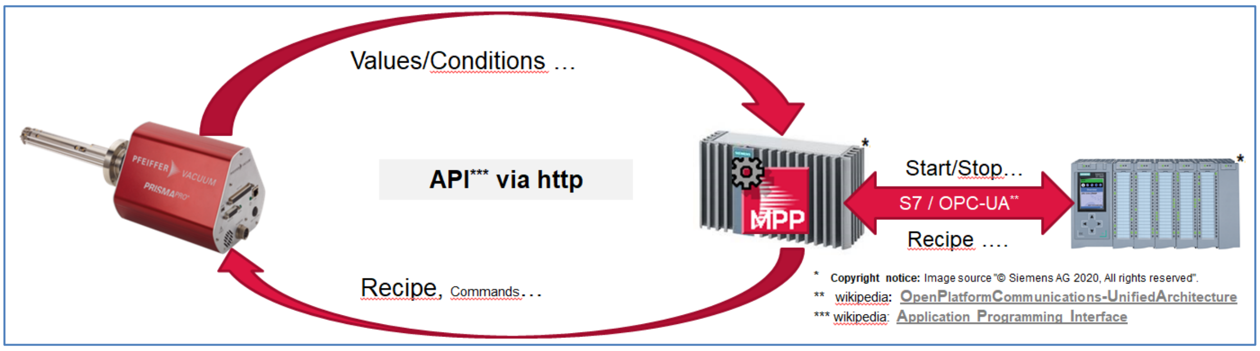
SYSTEM STATUS

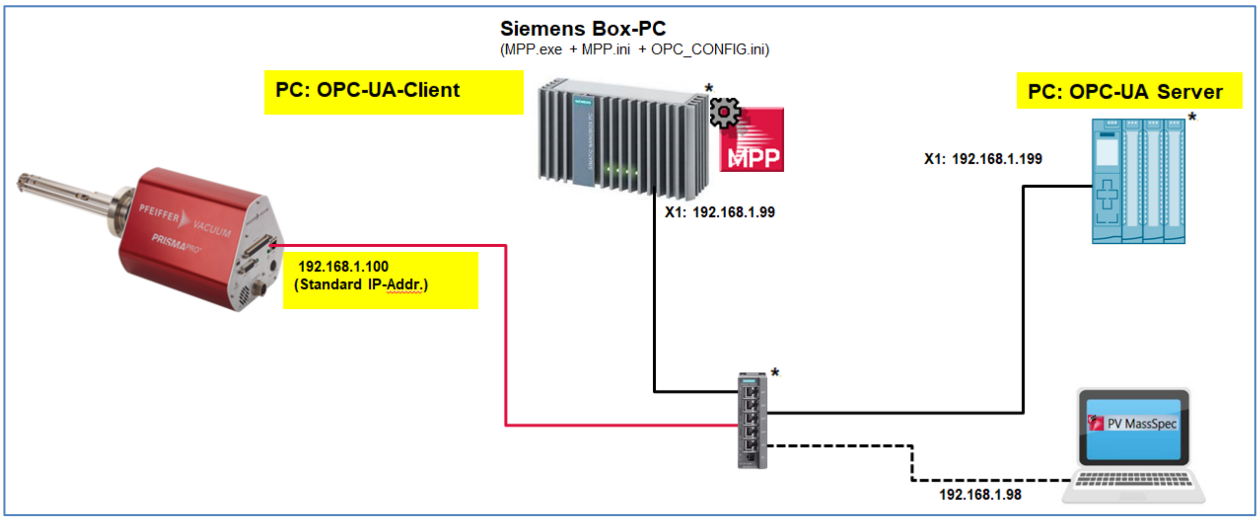
User	
No Security	None

RGA PrismaPro 200 Not Set.192.168.1.100 Position 1	STOPPED	Emission	NO Multiplier	Start	Configuration	Vacuum Diagnostics	Leak Check	Tune	Sensor Maintenance
RGA MPA 100 44525811 Position 2	NO SENSOR HEAD	Emission	Multiplier	Start	Configuration	Vacuum Diagnostics	Leak Check	Tune	Sensor Maintenance

3. MPP-Application:

Communication with Siemens PLC or OPC-UA-Server.


Values/Conditions …
API*** via http
Recipe, Commands…
MPP
Start/Stop…
S7 / OPC-UA**
Recipe ….
* Copyright notice: Image source "© Siemens AG 2020, All rights reserved".
** wikipedia: OpenPlatformCommunications-UnifiedArchitecture
*** wikipedia: Application Programming Interface



Further information about the MPP application can be found in our PV-Cloud: