

Document history

Version	Date	Major changes
1.2.	25.05.2020	General description incl. Example 004-ICS.py
1.1	08.05.2020	Document all methods for the singles data points
1.0	30.11.2019	Relayouted version

Disclaimer / Copyright

Copyright © 2020 by iseg Spezialelektronik GmbH / Germany. All Rights Reserved.

This document is under copyright of iseg Spezialelektronik GmbH, Germany. It is forbidden to copy, extract parts, duplicate for any kind of publication without a written permission of iseg Spezialelektronik GmbH. This information has been prepared for assisting operation and maintenance personnel to enable efficient use.

The information in this manual is subject to change without notice. We take no responsibility for any mistake in the document. We reserve the right to make changes in the product design without reservation and without notification to the users. We decline all responsibility for damages and injuries caused by an improper use of the device.

Safety

This section contains important security information for the installation and operation of the device. Failure to follow safety instructions and warnings can result in serious injury or death and property damage.

Safety and operating instructions must be read carefully before starting any operation.

We decline all responsibility for damages and injuries caused which may arise from improper use of our equipment.

Depiction of the safety instructions

DANGER!	
 DANGER!	<p>“Danger!” indicates a severe injury hazard. The non-observance of safety instructions marked as “Danger!” will lead to possible injury or death.</p>
WARNING!	
 WARNING!	<p>“Warning!” indicates an injury hazard. The non-observance of safety instructions marked as “Warning!” could lead to possible injury or death.</p>
CAUTION!	
 CAUTION!	<p>Advices marked as “Caution!” describe actions to avoid possible damages to property.</p>
INFORMATION	
 INFORMATION	<p>Advices marked as “Information” give important information.</p>



Read the manual.



HIGH VOLTAGE

Attention high voltage!



Important information.

Intended Use

The device may only be operated within the limits specified in the data sheet. The permissible ambient conditions (temperature, humidity) must be observed. The device is designed exclusively to control high voltage systems as specified in the data sheet.

It must only be used specified in Technical data. Any other use not specified by the manufacturer is not intended.

The manufacturer is not liable for any damage resulting from improper use.

Qualification of personnel

A qualified person is someone who is able to assess the work assigned to him, recognize possible dangers and take suitable safety measures on the basis of his technical training, his knowledge and experience as well as his knowledge of the relevant regulations.

General safety instructions

- Observe the valid regulations for accident prevention and environmental protection.
- Observe the safety regulations of the country in which the product is used.
- Observe the technical data and environmental conditions specified in the product documentation.
- You may only put the product into operation after it has been established that the high-voltage device complies with the country-specific regulations, safety regulations and standards of the application.
- The high-voltage power supply unit may only be installed by qualified personnel.

Important safety instructions

DANGER!



DANGER!

This device is part of a high voltage supplying systems.
High voltages are dangerous and may be fatal.

USE CAUTION WHILE WORKING WITH THIS EQUIPMENT.
BE AWARE OF ELECTRICAL HAZARDS.

Always follow at the minimum these provisions:

- High voltages must always be grounded
- Do not touch wiring or connectors without securing
- Never remove covers or equipment
- Always observe humidity conditions
- Service must be done by qualified personnel only

WARNING!



WARNING!

RAMP DOWN VOLTAGES!

Before insertion or removal of crate controller, please make sure, that all voltages are ramped down, modules are switched off and power cord is disconnected.

CAUTION!



CAUTION!

When controlling, with software, the high voltage systems, make sure that nobody is near the high voltage or can be injured.

INFORMATION



INFORMATION

Please check the compatibility with the devices used.

Table of Contents

Document history	2
Disclaimer / Copyright	2
Safety	3
Depiction of the safety instructions	3
Intended Use	4
Qualification of personnel	4
General safety instructions	4
Important safety instructions	5
1 General description	7
1.1 Configuration per WEB-socket	7
1.2 Configuration per WEB-browser	11
2 Importing the icsClientPython3 Python Module	14
2.1 Syntax	14
2.2 Help	14
2.3 Objects	14
2.3.1 The icsClient Object	14
2.3.2 The Connection Class	15
2.3.3 The crate class	17
2.3.4 The module class	21
2.3.5 The Channel Class	27
3 Appendix	39
4 Glossary	40
5 Manufacturer contact	41

1 General description

The icsClientPython3 library is part of iseg Communication Server iCS2 and can be used to automatize certain tasks like a data logging, process flows, event handling and so on. This library runs basically on different PC systems like Apple OS, Linux and Windows in connection with the icsService; but can also run directly on CC24, iCSmini2 and SHR hardware. The library offers most of the iCS items as python class structures for easy integration your own scripts. Therefore iCS2 Linux image contains some common python3 modules located under: `/usr/lib/python3.5`

Additional modules can be installed later with the python package installer PIP.

Python scripts can be configured by means of a WEB client via the iCSservice per WEB socket or by a WEB browser.

1.1 Configuration per WEB-socket

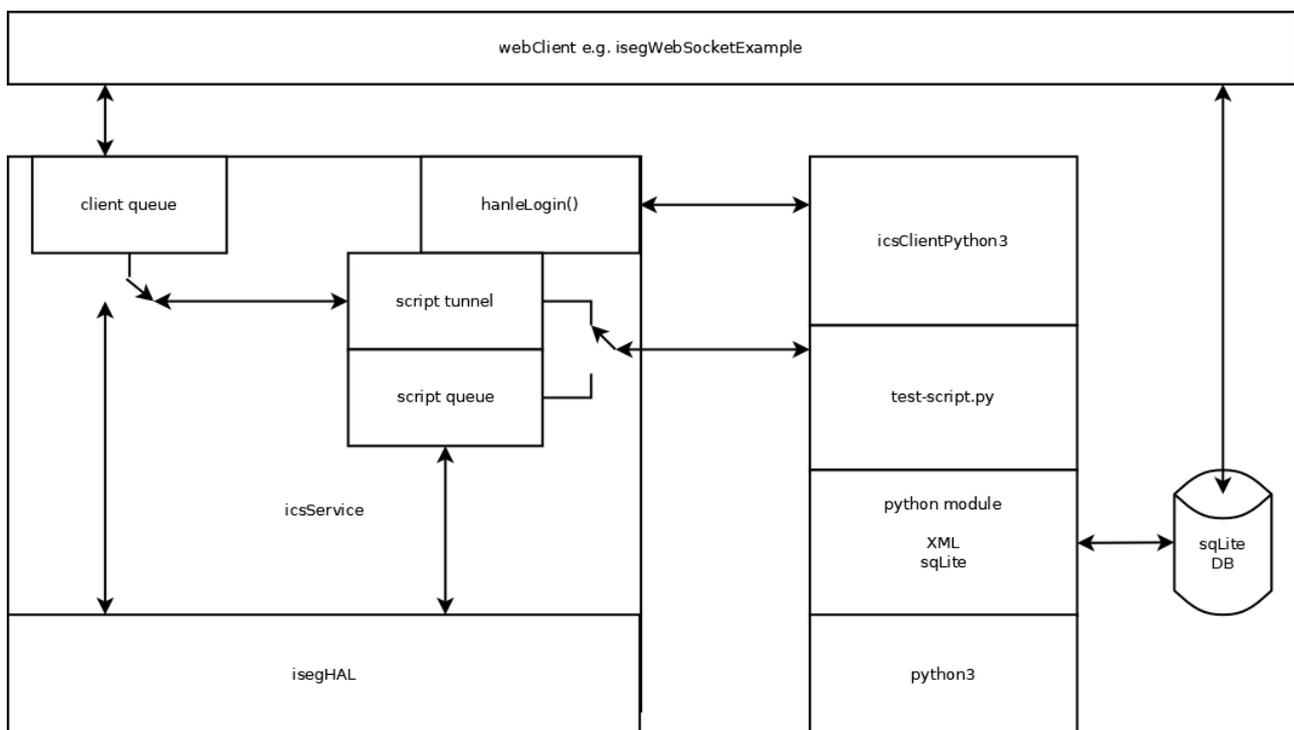


Figure 1

The iCS JSON frames in order to transmit introductions between iCS client and a script:

```
startScript
stopScript
autoScript
scriptStatus
scriptCommand
availableScripts
runningScripts
scriptData
```

Client starts a script:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": {
    "c": "startScript",
    "n": "ics-isolation-test.py",
    "t": ""
  },
  "r": "websocket"
}
```

Reply by the script:

```
{
  "i": "session_id_client",
  "t": "script",
  "c": [{
    "e": "scriptStatus",
    "d": "running", // running | stopped
    "n": "ics-isolation-test.py",
    "i": "session_id_script_123456"
  }
]
}
```

Client stops a script:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": {
    "c": "stopScript",
    "n": "ics-isolation-test.py",
    "i": "session_id_script_123456"
  },
  "r": "websocket"
}
```

Client kills a script:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": {
    "c": "killScript",
    "n": "ics-isolation-test.py",
    "i": "session_id_script_123456"
  },
  "r": "websocket"
}
```

Client configure a script to runs automatically after a reboot:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": {
    "c": "autoScript",
    "n": "iCSPythonDataLogger.py",
    "v": "enable/disable"
  },
  "r": "websocket"
}
```

Reply by the script:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": [{
    "e": "autoScripts",
    "d": ["iCSPythonDataLogger.py", "iCSPythonWalk.py"]
  }
],
  "r": "websocket"
}
```

Request script status:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": {
    "c": "scriptStatus",
    "i": "session_id_script_123456",
  },
  "r": "websocket"
}
```

Command from client to a script:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": {
    "c": "scriptCommand",
    "i": "session_id_script_123456",
    "d": { *** }
  },
  "r": "websocket"
}
```

Request available scripts:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": {
    "c": "scriptCommand",
    "i": "session_id_script_123456",
    "d": { *** }
  },
  "r": "websocket"
}
```

Reply by iCSservice:

```
{
  "i": "<session_id_client>",
  "t": "script",
  "c": [{
    "e": "availableScripts",
    "d": [
      {
        "name": "iCS-isolation-test.py",
        "status": "stopped",
        "autostart": "true",
        "hostIds": []
      },
      {
        "name": "iCS-test123.py",
        "status": "stopped",
        "autostart": "true",
        "hostIds": []
      }
    ]
  }
]}
}
```

Request started scripts:

```
{  
  "i": "<session_id_client>",  
  "t": "script",  
  "c": {  
    "c": "runningScripts"  
  },  
  "r": "websocket"  
}
```

Reply by iCSservice:

```
{  
  "i": "<session_id_client>",  
  "t": "script",  
  "c": [{  
    "e": "runningScripts",  
    "d": [{  
      "name": "iCS2-isolation-test.py",  
      "status": "running",  
      "autostart": "false",  
      "hostIds: ["123456-01","123456-02"]  
    }]  
  }]  
}
```

1.2 Configuration per WEB-browser

Python scripts can be uploaded to the iCS2 hardware by means of the WEB browser using ICSconfig > custom scripts > Script import. A couple of python scripts are preinstalled on the iCS2 hardware. These scripts and also your own future scripts can be configured here:

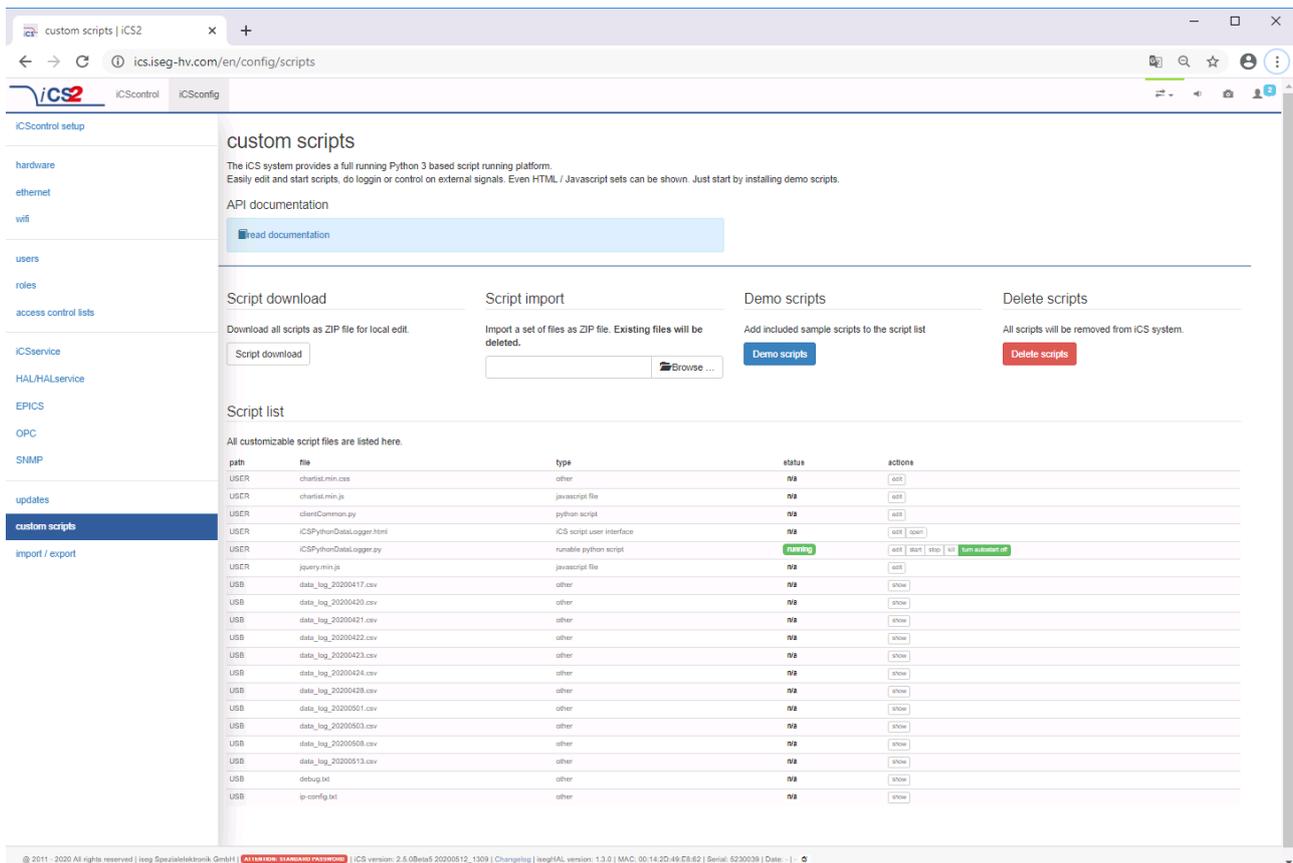


Figure 2

Custom scripts can be configured for an automatic start in connection with the icsConfig.xml after the next reboot. The following snippet from the icsConfig.xml shows the script node to store configuration settings:

```
<scripts>
  <script autoStart="true" file="iCSPythonDataLogger.py" group=""/>
</scripts>
```

```
#####
## Example 004-iCS.py outputs some data of an iseg High Voltage or W-IE-NE-R Low Voltage module in order to start
## writing your your own scripts. The module have to be located on line 0 and slot or device address 0.
#####
import argparse
import time
import traceback
import importlib
import types
import os

import clientCommon

connectionName = "new_iCS_connection"
connectionParameters = "ws://localhost"
connectionLogin = "admin"
connectionPassword = "password"

## import icsClient module
icsClient = clientCommon.importFromPath("/usr/lib", "libicsClientPython3")
if icsClient == 0:
    print("Failed to import icsClient module => Exit.")
    exit(1)

## establish a connection and wait until it is ready (with timeout)
connection = clientCommon.establishConnection(icsClient, connectionName, connectionParameters, connectionLogin,
                                              connectionPassword)
if type(connection) != icsClient.connection:
    print("Failed to connect => Exit.")
    exit(1)

## Power on CC24 crate
power_frame = "{\i:\<session_id>\, \t:\request\, \c\:[\c\:\getItem\, \p\:\p\:\
{\i:\:0\, \a\:\1000\, \c\:\}, \i\:\Control.power\, \v\:\, \u\:\}], \r\:\websocket\}"

connection.rawCommand(power_frame)
print(power_frame)

result = connection.rawCommand("{\i:\<session_id>\, \t:\request\, \c\:[\c\:\setItem\, \p\:\p\:\
{\i:\:0\, \a\:\1000\, \c\:\}, \i\:\Control.power\, \v\:\1\, \u\:\}], \r\:\websocket\}")
print(result)

all_crates = connection.getCrates()
print("All crates {0}.".format(all_crates))

all_modules = connection.getModules()
print("All modules {0}.".format(all_modules))

all_channels = all_modules[0].channels
print("All channels {0}.".format(all_channels))
```

```
##time to fill the cache from iCSservice
time.sleep(5)

def outputModule(item, function):
    print(item + str(function))

def outputModuleInt(item, function):
    print(item + str(int(function)))

outputModule("FirmwareName: ", all_modules[0].getStatusFirmwareName())
outputModuleInt("Seriennummer: ", all_modules[0].getStatusSerialNumber())
outputModule("Release: ", all_modules[0].getStatusFirmwareRelease())
outputModule("Temperatur: ", all_modules[0].getStatusTemperature())
#all_channels[1].setControlVoltageSet("123")

def output(channel, item, function):
    print(item + str(channel) + ": " + str(function))

def outputChannellItems():
    all_channels = all_modules[0].channels
    for channel in range(0, len(all_channels)):
        output(channel, "Status.voltageNominal", all_channels[channel].getStatusVoltageNominal())
    for channel in range(0, len(all_channels)):
        output(channel, "Control.voltageSet", all_channels[channel].getControlVoltageSet())
    for channel in range(0, len(all_channels)):
        output(channel, "Status.voltageMeasure", all_channels[channel].getStatusVoltageMeasure())
    for channel in range(0, len(all_channels)):
        output(channel, "Status.currentNominal", all_channels[channel].getStatusCurrentNominal())
    for channel in range(0, len(all_channels)):
        output(channel, "Control.currentSet", all_channels[channel].getControlCurrentSet())
    for channel in range(0, len(all_channels)):
        output(channel, "Status.currentMeasure", all_channels[channel].getStatusCurrentMeasure())
    for channel in range(0, len(all_channels)):
        output(channel, "Status.voltageLimitExceeded", all_channels[channel].getStatusVoltageLimitExceeded())
    for channel in range(0, len(all_channels)):
        output(channel, "Status.currentLimitExceeded", all_channels[channel].getStatusCurrentLimitExceeded())
    for channel in range(0, len(all_channels)):
        output(channel, "Status.currentTrip", all_channels[channel].getStatusTrip())
    for channel in range(0, len(all_channels)):
        output(channel, "Status.on", all_channels[channel].getStatusOn())
    for channel in range(0, len(all_channels)):
        output(channel, "Status.emergenc", all_channels[channel].getStatusEmergency())
    for channel in range(0, len(all_channels)):
        output(channel, "Setup.delayedTripAction", all_channels[channel].getSetupDelayedTripAction())
    for channel in range(0, len(all_channels)):
        output(channel, "Setup.delayedTripTime", all_channels[channel].getSetupDelayedTripTime())

outputChannellItems()

clientCommon.disconnectAll(icsClient)
```

2 Importing the icsClientPython3 Python Module

2.1 Syntax

The python module can be imported using the command

```
> icsClient = importlib.import_module("icsClientPython3")
```

In order to find the Python module the "sys.path" variable must include the path to icsClientPython3.pyd (Windows), icsClientPython3.so (Linux) or "icsClientPython3.dylib" (macOs).

As the icsClientPython3 module depends on other shared libraries, the LD_LIBRARY_PATH variable might have to be adjusted on Linux.

The returned class object can be used for connecting and communicating with icsService.

2.2 Help

```
> help(icsClient)
```

The following user guide gives a short overview over the most important functions, classes and methods.

2.3 Objects

2.3.1 The icsClient Object

The icsClient Object defines a number of methods:

- the method connect(name, ip_address, login, password) can be used to establish the connection to an icsService.
- getConnection(): This function retrieves the list of open icsService connections. The objects in the list are "connection class objects" as described below.

2.3.2 The Connection Class

The connection objects are instantiated by the icsClient object. A list of connection objects can be retrieved by the method icsClient.getConnections(). The following main methods can be used on a connection object:

- disconnect(): this method closes the connection to an icsService.
- getCrates(): this method returns a list of crate objects controlled by the current connection. The crate class is defined below.
- getModules(): this method returns a list of module objects controlled by the current connection. The module class is defined below.
- getStatusConnectionName(): this method returns the connection's name.
- getStatusConnectionStatus(): this method returns the connection's status as string.
- rawCommand(command): this method sends a command to an icsService connection. The command has to be a JSON string.
- setControlDisconnect(): this method disconnects from the icsService.

The first output lines from the example 004-ics.py demonstrate the connection process:

```
Successfully imported module "libicsClientPython3" (version 20.2.11.0) from path "/usr/lib"
Waiting for connection to be created ...
Waiting for connection to ws://localhost to be ready ...
1 sec : connection status == Configured
Connected to "new_ics_connection"
{"i":"<session_id>","t":"request","c":[{"c":"getItem","p":{"p":
{"l":"0","a":"1000","c":""},"i":"Control.power","v":"","u":""}}],"r":"websocket"}
ok
All crates [0.1000].
All modules [0.1].
All channels [0.1.0, 0.1.1, 0.1.1, 0.1.2, 0.1.3, 0.1.4, 0.1.5, 0.1.6, 0.1.7, 0.1.8, 0.1.9, 0.1.10, 0.1.11,
0.1.12, 0.1.13, 0.1.14, 0.1.15].
```

- getControlLiveInsertion(): switch live insertion mode on or off
- getSetupCycleTime(): cycletime (update rate) in ms
- getStatusCanState(): CAN bus state
- getStatusCanText(): CAN bus state text information
- getStatusConnectedClients(): amount of clients connected with iCS
- getStatusDiscover(): comma separated interface list
- getStatusEthernetState(): state of ethernet interface
- getStatusHeartBeat(): isegHAL cycle counter as heart beat
- getStatusLiveInsertion(): state of live insertion mode
- getStatusProtocols(): comma separated interface protocol list
- getStatusRunningStateEthernet(): running state of ethernet adapter
- getStatusRunningStateIcs(): general state of iCS system
- getStatusRunningStateWeb(): running state of webserver
- getStatusRunningStateWifi(): running state of Wifi connection
- getStatusSessionId(): session identifier
- getStatusUsbAuxState(): state of AUX USB interface
- getStatusUsbAuxText(): AUX USB state text information
- getStatusUsbRemoteState(): state of USB remote interface
- getStatusUsbRemoteText(): USB remote state text information
- getStatusWifiState(): state of Wifi interface
- rawCommand(): This method sends a command to an icsService connection. The command has to be a JSON string.
- SendMessage():
- sendMessageList():
- setControlAutoConfig(): automatic iCS system configuration
- setControlConnect(): connect an interface
- setControlDisconnect(...): disconnect an interface
- setControlLiveInsertion(): switch live insertion mode on or off
- setControlReboot(): reboot iCS Server
- setControlRestartScan(): rescan all connected devices (modules, crates)
- setControlServiceRestart(): restart one of the available interface services
- setControlServiceStart(): start one of the available interface services
- setControlServiceStop(): stop one of the available interface services
- setControlShutdown(): shutdown iCS Server

- `setControlWifi()`: switch wifi on or off
- `setSetupCycleTime()`: `cycletime (updaterate)` in ms

Data descriptors defined here:

- `crates`: crates managed by connection
- `modules`: modules managed by connection
- `name`: connection name

The complete list of methods can be obtained by using the built-in help functions

> **`help(icsClient.connection)`**

2.3.3 The crate class

A list of crate objects can be obtained using the `connection.getCrates()` method.

The methods defined here:

__new__(*args, **kwargs) from builtins.type

Description: Create and return a new object. See `help(type)` for accurate signature.

getControlPower(...)

Description: Returns true if the crate is switched on.

getErrorBatteryNotGood(...)

Description: The battery voltage is or was out of valid range.

getErrorMainsNotGood(...)

Description: Mains voltage is or was out of valid range.

getErrorSupplyNotGood(...)

Description: At least one supply voltage is or was out of valid range.

getErrorTemperatureNotGood(...)

Description: Crate temperature has been or is out of valid range.

getEventBatteryNotGood(...)

Description: The battery voltage is or was out of valid range.

getEventMainsNotGood()

Description: Mains voltage is or was out of valid range.

getEventSupplyNotGood(...)

Description: At least one supply voltage is or was out of valid range.

getEventTemperatureNotGood(...)

Description: Crate temperature has been or is out of valid range.

getStatusBatteryGood(...)

Description: The battery voltage is in valid range.

getStatusErrorCounter(...)

Description: Amount of errors occurred in the crate and its modules and corresponding channels.

getStatusEventCounter(...)

Description: Amount of events occurred in the crate and its modules and corresponding channels.

getStatusFirmwareName(...)

Description: name of firmware

getStatusFirmwareRelease(...)

Description: firmware version

getStatusIsAlive(...)

Description: The crate is alive (in standby or powered on).

getStatusMainsGood(...)

Description: Mains voltage is in valid range.

getStatusOn(...)

Description: At least one channel is active or has measured voltage above 63 Volts.

getStatusPower(...)

Description: The crate is switched on (no standby).

getStatusRunningState(...)

Description crate status:

- error - at least one error in crate or its modules or channels
- info - at least one channel ramping
- ok - at least one channel at HV-ON
- off - all channels at HV-OFF, disabled - crate not available

getStatusSupplyGood(...)

Description: All supply voltages are in valid range.

getStatusTemperature0(...)

Description: Measured temperature in celsius degree.

getStatusTemperature1(...)

Description: Measured temperature in celsius degree.

getStatusTemperatureGood(...)

Description: Crate temperature is in valid range.

setControlClearAll(...)

Description: Clear all events and errors of the crate, even the module and channel errors and events of this crate.

setControlClearErrors(...)

Description: Clear all errors of the crate, even the module and channel errors of this crate.

setControlClearEvents(...)

Description: Clear all events of the crate, even the module and channel events of this crate.

setControlInstallFirmware(...)

Description: Update crate controller firmware with given filename.

setControlPower(...)

Description: Switches the crate on or off.

Data descriptors defined here:

address

Description: crate address

connection

Description: connection name

line

Description: line index

module

Description: module address

The complete list of methods can be obtained by using the built-in help functions

> **help(icsClient.crate)**

2.3.4 The module class

A list of module objects can be obtained using the connection.getModules() method. The most important module methods are the following:

__new__(*args, **kwargs) from builtins.type

Description: Create and return a new object. See help(type) for accurate signature.

GetControlAdjustment(...)

Description: activate / deactivate fine adjustment

getControlCurrentRampspeed(...)

Description: rampspeed of current in %/s

getControlKill(...)

Description: activate / deactivate KILL mode

getControlModuleChannelEventMask(...)

Description: bitmask to mask/unmask channel events at module level

getControlModuleEventMask(...)

Description: bitmask to mask/unmask module events

getControlUserHardwareCheck(...)

Description: only for iseq MICC-systems: user confirms that MICC configuration has been verified

getControlVoltageRampspeed(...)

Description: rampspeed of voltage in %/s

getErrorInterlockOut(...)

Description: caught InterlockOut event

getErrorNeedService(...)

Description: hardware failure detected, vendor support needed

getErrorNeedUserHardwareCheck(...)

Description: only for iseq MICC-systems: user has to confirm that MICC configuration has been verified

getErrorSafetyLoopWasOpened(...)

Description: Safety-Loop has been opened

getErrorSupplyNotGood(...)

Description: at least one supply voltage is or was out of valid range

getErrorTemperatureNotGood(...)

Description: board temperature has been or is out of valid range

getEventInterlockOut(...)

Description: caught InterlockOut event

getEventSafetyLoopWasOpened(...)

Description: Safety-Loop has been opened

getEventSupplyNotGood(...)

Description: at least one supply voltage is or was out of valid range

getEventTemperatureNotGood(...)

Description: board temperature has been or is out of valid range

getSetupAdcSampleRate(...)

Description: Samplerate of ADC for measured voltage and current

getSetupDigitalFilter(...)

Description: moving average on specified steps of samples

getStatusAdjustment(...)

Description: fine adjustment is activated

getStatusErrorCounter(...)

Description: amount of errors occurred in module and corresponding channels

getStatusEventCounter(...)

Description: amount of events occurred in module and corresponding channels

getStatusFirmwareName(...)

Description: name of firmware

getStatusFirmwareRelease(...)

Description: firmware version

getStatusInterlockOut(...)

Description: InterlockOut active

getStatusIsAlive(...)

Description: module is powered on and able to communicate

getStatusNeedService(...)

Description: hardware failure detected, vendor support needed

getStatusRunningState(...)

Description: module status:
error – at least one error in module or channel
info – at least one channel ramping
ok – at least one channel at HV-ON
off – all channels at HV-OFF
disabled – module not available

getStatusSafetyLoopClosed(...)

Description: Safety-Loop is closed

getStatusSerialNumber(...)

Description: serialnumber of the module

getStatusSupplyGood(...)

Description: all supply voltages are in valid range

getStatusTemperature(...)

Description: measured temperature in celsius degree

getStatusTemperatureGood(...)

Description: board temperature is in valid range

setControlAdjustment(...)

Description: activate / deactivate fine adjustment

setControlClearAll(...)

Description: clear all events and errors of the module, even the channel errors and channel events of the module

setControlClearErrors(...)

Description: clear all errors of the module, even the channel errors of this module

setControlClearEvents(...)

Description: clear all events of the module, even the channel events of this module

setControlCurrentRampspeed(...)

setControlInstallFirmware(...)

Description: update module firmware with given filename

setControlKill(...)

Description: activate / deactivate KILL mode

setControlModuleChannelEventMask(...)

Description: bitmask to mask/unmask channel events at module level

setControlModuleEventMask(...)

Description: bitmask to mask/unmask module events

setControlUserHardwareCheck(...)

Description: only for iseq MICC-systems: user confirms that MICC configuration has been verified

setControlVoltageRampspeed(...)

Description: rampspeed of voltage in %/s

setSetupAdcSampleRate(...)

Description: Samplerate of ADC for measured voltage and current

setSetupDigitalFilter(...)

Description: moving average on specified steps of samples

Data descriptors defined here:

address

Description: module address

channels

Description: module channels

connection

Description: connection name

line

Description: line index

The module's channels can be stored in a list of channel objects. This list is a member of the module. It can be retrieved as `module.channels`. The channel class is described below.

The complete list of methods can be obtained by using the built-in help functions

> **help(icsClient.module)**

2.3.5 The Channel Class

A list of channel objects can be obtained by reading the `module.channels` member. The most important channel methods are the following:

`__new__(*args, **kwargs) from builtins.type`

Description: Create and return a new object. See `help(type)` for accurate signature.

`getControlChannelEventMask(...)`

bitmask to mask/unmask channel events

`getControlCurrentBounds(...)`

Description: current tolerance-bounds value relative (+/-) to current set

`getControlCurrentRampspeedDown(...)`

Description: rampspeed to decrease the current in mA/s

`getControlCurrentRampspeedUp(...)`

rampspeed to increase the current in mA/s

`getControlCurrentSet(...)`

Description: current set value

`getControlEmergency(...)`

Description: 1 = switch channel without ramp to "Emergency Off", turn-on-lock is active;
0 = exit "Emergency Off"

`getControlOn(...)`

Description: 1 = turn on channel with ramp;
0 = turn off channel with ramp

`getControlVctCoefficient(...)`

Description: VCT coefficient

`getControlVoltageBounds(...)`

Description: voltage tolerance-bounds value relative (+/-) to voltage set

getControlVoltageRampSpeedDown(...)

Description: rampspeed to decrease the voltage in V/s

getControlVoltageRampSpeedUp(...)

Description: rampspeed to increase the voltage in V/s

getControlVoltageSet(...)

Description: voltage set value

getErrorArc(...)

Description: maximum amount of allowed arcs reached

getErrorCurrentBoundExceeded(...)

Description: current tolerance range has been left (upper and/or lower)

getErrorCurrentLimitExceeded(...)

Description: current limit exceeded, HV turned off if KILL == 1

getErrorEmergency(...)

Description: "Emergency Off" has been active; HV has been turned off

getErrorExternalInhibitActive(...)

Description: Inhibit has been active; HV turned off

getErrorTrip(...)

Description: current trip limit exceeded, HV turned off

getErrorVoltageBoundExceeded(...)

Description: voltage tolerance range left (upper and/or lower)

getErrorVoltageLimitExceeded(...)

Description: voltage limit has been exceeded, HV turned off if KILL == 1

getEventArc(...)

Description: at least one arc has been detected

getEventArcError(...)

Description:

getEventConstantCurrent(...)

Description: channel has been or is in constant current regulation

getEventConstantPower(...)

Description: Has been or is in constant power regulation

getEventConstantVoltage(...)

Description: has been or is in constant voltage regulation

getEventCurrentBoundExceeded(...)

Description: current tolerance range has been left (upper and/or lower)

getEventCurrentLimitExceeded(...)

Description: current limit exceeded, HV turned off if KILL == 1

getEventCurrentRampDown(...)

Description: Current has been or is ramp down

getEventCurrentRampUp(...)

Description: Current has been or is ramp up

getEventEmergency(...)

Description: "Emergency Off" has been active; HV has been turned off

getEventEndOfCurrentRamp(...)

Description: current set value reached; ramping ends

getEventEndOfRamp(...)

Description: set value reached; ramping ends

getEventExternalInhibitActive(...)

Description: Inhibit has been active; HV turned off

getEventInputError(...)

Description: an input error will be generated after wrong or invalid data was sent

getEventMaxPower(...)

Description: Max power has been or is reached

getEventTrip(...)

Description: current trip limit exceeded, HV turned off

getEventVoltageBottom(...)

Description: Output has been set to voltage bottom after a trip

getEventVoltageBoundExceeded(...)

Description: voltage tolerance range has been left (upper and/or lower)

getEventVoltageBoundLower(...)

Description: HV has been or is lower than to the lower voltage bound

getEventVoltageBoundUpper(...)

Description: HV has been or is higher than the upper voltage bound

getEventVoltageLimitExceeded(...)

Description: voltage limit has been exceeded, HV turned off if KILL == 1

getEventVoltageRampDown(...)

Description: HV has been or is ramp down

getEventVoltageRampUp(...)

Description: HV has been or is ramp up

getSetupDelayedTripAction(...)

Description: action on a trip event:

- 0 = no action
- 1 = shut down voltage of the channel with ramp
- 2 = shut down voltage of the channel without ramp
- 3 = shut down the whole module without ramp
- 4 = disable the delayed trip action (no Event.trip generation)

getSetupDelayedTripTime(...)

Description: time in ms until delayedTripAction is called

getSetupExternalInhibitAction(...)

Description: action on external inhibit active:

- 0 = no action
- 1 = ramp down voltage of this channel
- 2 = shut down high voltage of the channel without ramp
- 3 = shut down the whole module without ramp

getSetupOutputMode(...)

Description: number of HV mode, option switchable HV-generation modes

getSetupOutputPolarity(...)

Description: polarity, option switchable polarity

getSetupResistanceExternal(...)

Description: resistance external, option stacked output channels

getSetupVoltageBottom(...)

Description: reduce output voltage to the percentage value = $p * \text{voltageSet} / 100$, option stacked output channels

getSetupVoltageRampPriority(...)

Description: priority controlled voltage ramp, option stacked output channels

getStatusArc(...)

Description: arc detected

getStatusConstantCurrent(...)

Description: is in constant current regulation

getStatusConstantPower(...)

Description: In constant power regulation

getStatusConstantVoltage(...)

Description: in constant voltage regulation

getStatusCurrentBoundExceeded(...)

Description: current tolerance range left (upper and/or lower)

getStatusCurrentLimit(...)

Description: $(\text{Module.llimit} * \text{Channel.lnominal})/100\%$ (Sliderlimit defined by hardware or software)

getStatusCurrentLimitExceeded(...)

Description: current limit exceeded

getStatusCurrentMeasure(...)

Description: measured current

getStatusCurrentMeasureWithTimestamp(...)

Description: measured current and timestamp

getStatusCurrentMode(...)

Description: current mode, option switchable HV-generation modes

getStatusCurrentModeList(...)

Description: current mode list, option switchable HV-generation modes

getStatusCurrentNominal(...)

Description: nominal current

getStatusCurrentRampDown(...)

Description: Current is ramping down with specified ramp

getStatusCurrentRampUp(...)

Description: Current is ramping up with specified ramp

getStatusCurrentRamping(...)

Description: Current is changing with specified ramp

getStatusCurrentRampspeedMax(...)

Description: maximum rampspeed to change the current in mA/s

getStatusCurrentRampspeedMin(...)

Description: minimum rampspeed to change the current in mA/s

getStatusEmergency(...)

Description: "Emergency Off" is active

getStatusErrorCounter(...)

Description: amount of occurred channel errors

getStatusEventCounter(...)

Description: amount of occurred channel events

getStatusExternalInhibitActive(...)

Description: Inhibit active

getStatusLowCurrentRange(...)

Description: Current is measured in 2nd Current Range

getStatusMaxPower(...)

Description: Max power is reached

getStatusOn(...)

Description: HV is turned on

getStatusRamping(...)

Description: HV is changing with specified ramp

getStatusRunningState(...)

Description: channel state:
error – channel has an error
info – channel is ramping
ok – HV is on
off – HV is off
disabled – channel not available

getStatusTemperatureExternal(...)

Description: temperature of external sensor

getStatusTrip(...)

Description: current trip limit exceeded

getStatusVoltageBoundExceeded(...)

Description: voltage tolerance range left (upper and/or lower)

getStatusVoltageBoundLower(...)

Description: HV is lower than the lower voltage bound

getStatusVoltageBoundUpper(...)

Description: HV is higher than the upper voltage bound

getStatusVoltageLimit(...)

Description: $(\text{Module.Vlimit} * \text{Channel.Vnominal})/100\%$ (Sliderlimit, defined by Hardware or Software)

getStatusVoltageLimitExceeded(...)

Description: voltage limit exceeded

getStatusVoltageMeasure(...)

Description: measured voltage

getStatusVoltageMeasureWithTimestamp(...)

Description: measured voltage and timestamp

getStatusVoltageMode(...)

Description: voltage mode, option switchable HV-generation modes

getStatusVoltageModeList(...)

Description: voltage mode list, option switchable HV-generation modes

getStatusVoltageNominal(...)

Description: nominal voltage

getStatusVoltageRampDown(...)

Description: HV is ramping down with specified ramp

getStatusVoltageRampUp(...)

Description: HV is ramping up with specified ramp

getStatusVoltageRamping(...)

Description: HV is changing with specified ramp

getStatusVoltageRampspeedMax(...)

Description: maximum rampspeed to change the voltage in V/s

getStatusVoltageRampspeedMin(...)

Description: minimum rampspeed to change the voltage in V/s

setControlChannelEventMask(...)

Description: bitmask to mask/unmask channel events

setControlClearAll(...)

Description: clear all channel events and channel errors

setControlClearErrors(...)

Description: clear all channel errors

setControlClearEvents(...)

Description: clear all channel events

setControlCurrentBounds(...)

Description: current tolerance-bounds value relative (+/-) to current set

setControlCurrentRampspeedDown(...)

Description: rampspeed to decrease the current in mA/s

setControlCurrentRampspeedUp(...)

Description: rampspeed to increase the current in mA/s

setControlCurrentSet(...)

Description: current set value

setControlEmergency(...)

Description: 1 = switch channel without ramp to "Emergency Off", turn-on-lock is active;
0 = exit "Emergency Off"

setControlOn(...)

Description: 1 = turn on channel with ramp
0 = turn off channel with ramp

setControlVctCoefficient(...)

Description: VCT coefficient

setControlVoltageBounds(...)

Description: voltage tolerance-bounds value relative (+/-) to voltage set

setControlVoltageRampspeedDown(...)

Description: rampspeed to decrease the voltage in V/s

setControlVoltageRampSpeedUp(...)

Description: ramp speed to increase the voltage in V/s

setControlVoltageSet(...)

Description: voltage set value

setSetupDelayedTripAction(...)

Description: action on a trip event:

- 0 = no action
- 1 = shut down voltage of the channel with ramp
- 2 = shut down voltage of the channel without ramp
- 3 = shut down the whole module without ramp
- 4 = disable the delayed trip action (no Event.trip generation)

setSetupDelayedTripTime(...)

Description: time in ms until delayedTripAction is called

setSetupExternalInhibitAction(...)

Description: action on external inhibit active:

- 0 = no action
- 1 = ramp down voltage of this channel
- 2 = shut down high voltage of the channel without ramp
- 3 = shut down the whole module without ramp

setSetupOutputMode(...)

Description: number of HV mode, option switchable HV-generation modes

setSetupOutputPolarity(...)

Description: polarity, option switchable polarity

setSetupResistanceExternal(...)

Description: resistance external, option stacked output channels

setSetupVoltageBottom(...)

Description: reduce output voltage to the percentage value = $p * \text{voltageSet} / 100$, option stacked output channels

setSetupVoltageRampPriority(...)

Description: priority controlled voltage ramp, option stacked output channels

setStatusCurrentRampspeedMax(...)

Description: maximum rampspeed to change the current in mA/s

setStatusCurrentRampspeedMin(...)

Description: minimum rampspeed to change the current in mA/s

setStatusVoltageRampspeedMax(...)

Description: maximum rampspeed to change the voltage in V/s

setStatusVoltageRampspeedMin(...)

Description: minimum rampspeed to change the voltage in V/s

Data descriptors defined here:

connection

Description: connection name

index

Description: channel index

line

Description: line index

module

Description: module address

The complete list of methods can be obtained by using the built-in help functions

> help(icsClient.channel)

3 Appendix

For more information please use the following download links:

This document
ics-python3.pdf
CAN EDCP Programmers-Guide
http://download.iseq-hv.com/SYSTEMS/MMS/CAN_EDCP_Programmers-Guide.pdf
iseq Hardware Abstraction Layer
http://download.iseq-hv.com/SYSTEMS/MMS/iseqHardwareAbstractionLayer.pdf

4 Glossary

SHORTCUT	MEANING
V_{nom}	nominal output voltage
V_{out}	output voltage
V_{set}	set value of output voltage
V_{mon}	monitor voltage
V_{meas}	digital measured value of voltage
V_{p-p}	peak to peak ripple voltage
V_{in}	input / supply voltage
V_{type}	type of output voltage (AC, DC)
V_{ref}	internal reference voltage
V_{max}	limit (max.) value of output voltage
$\Delta V_{out} - [\Delta V_{in}]$	deviation of V_{out} dep. on variation of supply voltage
$\Delta V_{out} - [\Delta R_{load}]$	deviation of V_{out} dep. on variation of output load
V_{bounds}	Voltage bounds, a tolerance tube $V_{set} \pm V_{bounds}$ around V_{set}
I_{nom}	nominal output current
I_{out}	output current
I_{set}	set value of output current
I_{mon}	monitor voltage of output current
I_{meas}	digital measured value of current
I_{trip}	current limit to shut down the output voltage
I_{in}	input / supply current
I_{max}	limit (max.) value of output current
I_{limit}	Current Limit.
I_{bounds}	Current bounds, a tolerance tube $I_{set} \pm I_{bounds}$ around I_{set}
P_{nom}	nominal output power
P_{in}	input power
P_{in_nom}	nominal input power
T	temperature
T_{REF}	Reference temperature
ON	HV ON/OFF
/ON	HV OFF/ON
CH	channel(s)
HV	high voltage
LV	low voltage
GND	signal ground
INH	Inhibit
POL	Polarity
KILL	KillEnable

5 Manufacturer contact

iseq Spezialelektronik GmbH

Bautzner Landstr. 23

01454 Radeberg / OT Rossendorf

GERMANY

FON: +49 351 26996-0 FAX: +49 351 26996-21

www.iseq-hv.com | info@iseq-hv.de | sales@iseq-hv.de