SIEMENS

Simcenter™ Micred™ T3STER™ SI API Description

Software Version 2.2.0 Release 2301
February 2023

SIEMENS

About Siemens Digital Industries Software

Siemens Digital Industries Software is a leading global provider of product life cycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens Digital Industries Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens Digital Industries Software products and services, visit www.siemens.com/plm.

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Table of Contents

Chapter 1
Introduction

This document provides a description of the Simcenter™ Micred™ T3STER™ SI Application Programming Interface (API).

Note

Before starting to use the T3STER SI API, make sure that your API version matches the version described in this document. To check the version of your API, use the **[BE_VERSION]** command.

For more information, see Get Internal Build Version [BE_VERSION].

Communication is done through WebSocket in JSON format. In case an invalid JSON message is received, the system ignores it and returns an error message.

The JSON standard can be obtained form the following website:

https://www.json.org

You can access WebSocket through port 8085:

ws://<IP_address>:8085/

For example, if you access the Control Software through *http://192.168.0.123:8085/*, then you can connect to WebSocket at *ws://192.168.0.123:8085/*.

T3STER SI communication functions as a request–response protocol in the client–server model. That is, for each command received, the server will send one and only one answer message.

The T3STER SI serves only one WebSocket at a time (in a first come, first served mode). All other connected clients will receive the **"CONTROL_FORBIDDEN"** answer as a response to all commands.

The following is an example answer message in case control is forbidden, because another client is connected to the T3STER SI:

```
{
    "Answer":"CONTROL_FORBIDDEN"
}
```

All requests must contain the *Command* field. Each command and query may have other fields: mandatory or optional. If something is mandatory, that will not be described in this document. However, optional fields will be marked as optional.

Every request has an optional string type field, named *UniqueID*. If this field is present in the request, then it will be copied to the response to identify the answer.

Example command:

```
{
    "Command": "EXAMPLE_COMMAND",
    "UniqueID": "MyUniqueID123"
}
```

Answer message:

```
{
    "Answer": "EXAMPLE_ANSWER",
    "UniqueID": "MyUniqueID123"
}
```

All replies contain an *Answer* field. For example:

```
{
    "Answer": "OK"
}
```

The *Answer* is typically a short message in all capital letters. For most commands (not all), the reply will contain a *Message* field with some general information. For example, if an error occurs, the *Message* field contains the reason for the error.

Chapter 2
System-Wide Commands

This section provides a description of system-wide commands.

Powering

This section provides a description of the commands related to powering the T3STER SI system.

Shut Down the T3STER SI [SHUTDOWN]

This section provides a description of the **[SHUTDOWN]** command.

To shut down the T3STER SI, use the following command:

```
{
    "Command": "SHUTDOWN"
}
```

Answer message:

```
{
    "Answer": "OK"
}
```

When the answer message is sent back to the user, the system will shut down.

Reboot the T3STER SI [REBOOT]

This section provides a description of the **[REBOOT]** command.

To reboot the system, use the following command:

```
{
    "Command": "REBOOT"
}
```

Answer message:

```
{
    "Answer": "OK"
}
```

When the answer message is sent back to the user, the system will start the reboot process.

System Clock

This section provides a description of the commands related to the system clock.

Get the Current Date and Time [GET_DATETIME]

This section provides a description of the **[GET_DATETIME]** command.

To query the current date and time from the T3STER SI system, use the **[GET_DATETIME]** command.

The system sends back its time as a string in the 24-character long simplified extended ISO format (YYYY-MM-DDTHH:mm:ss.sssZ). The timezone is always zero UTC offset, as denoted by the suffix "Z".

Command:

```
{
    "Command": "GET_DATETIME"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "DateTime": "YYYY-MM-DDTHH:mm:ss.sssZ",
     "Message": ""
}
```

Answer message example:

```
{
     "Answer": "OK",
     "DateTime": "2022-02-02T12:13:14Z",
     "Message": ""
}
```

Set the current date and time [SET_DATETIME]

This section provides a description of the **[SET_DATETIME]** command.

To set the date and time in the system, use the **[SET_DATETIME]** command.

Only the 24-character long ISO format is accepted with zero UTC offset, which is the same as in the answer message to the **[GET_DATETIME]** command.

Command format:

```
{
    "Command": "SET_DATETIME",
    "DateTime": "YYYY-MM-DDTHH:mm:ss.sssZ"
}
```

Command example:

```
{
    "Command": "SET_DATETIME",
    "DateTime": "2022-02-02T12:13:14Z"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Back-End Version Queries

This section provides a description of the commands related to back-end version queries.

Get Full Software Release Version [PACKAGE_VERSION]

This section provides a description of the **[PACKAGE_VERSION]** command.

To obtain the version of the installed Control Software package, use the following query:

```
{
    "Command": "PACKAGE_VERSION"
}
```

Answer message format:

```
{
    "Answer": "SIEMENS_VERSION-INTERNAL_VERSION-BUILD_HASH"
}
```

Answer message example:

```
{
     "Answer": "2201-v1.0.0-g1a2b3c4d"
}
```

Get Internal Build Version [BE_VERSION]

This section provides a description of the **[BE_VERSION]** command.

To query the internal version of the installed Control Software package, use the following command:

```
{
    "Command": "BE_VERSION"
}
```

Answer message format:

```
{
    "Answer": "INTERNAL_VERSION-BUILD_HASH",
    "ApiVersion": "API_VERSION"
}
```

Answer message example:

```
{
    "Answer": "v2.1.0-g1a2b3c4d",
    "ApiVersion": "2.1.0"
}
```

Generate an Error Report [GENERATE_ERROR_REPORT]

This section provides a description of the **[GENERATE_ERROR_REPORT]** command.

If you detect any malfunction in the system, generate an event report and attach it to your bug report.

To generate an event report, use the following command:

```
{
    "Command": "GENERATE_ERROR_REPORT"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": "/measurement/log/
exported_log_2022_02_17_22_21_46_148.t3silog"
}
```

You can get the link to the event report from the *Message* field. To download the report, attach the link to the report to the base URL of your system.

For example, if you access the Control Software through *http://192.168.0.123:8085*, then the report in the example above can be downloaded from the following URL:

http://192.168.0.123:8085/measurement/log/exported_log_2022_02_17_22_21_46_148.t3silog

The report is encoded. Make sure that you do not edit the report after downloading it.

Query the System Integrity Status [QUERY_SYSTEM_INTEGRITY]

This section provides a description of the **[QUERY_SYSTEM_INTEGRITY]** command.

System integrity refers to the status of the whole system. All pieces of firmware and software must be compatible with one another, and the system must be in a stable state.

If there is any mismatch between software and firmware versions (for example, the application card firmware is outdated), or any malfunction is detected in the system (for example, an unexpected external hardware detach), then the system will reject any non-system-wide command.

To query the system integrity status, use the following command:

```
{
    "Command": "QUERY_SYSTEM_INTEGRITY"
}
```

If no issue is detected in the system, the answer will be **“OK”**:

```
{
     "Answer": "OK",
     "Message": ""
}
```

If any error is detected, the *Answer* field will contain the value **"SYSTEM_INTEGRITY_ERR"**. For example:

```
{
     "Answer": "SYSTEM_INTEGRITY_ERR",
    "Message": "Invalid system state, try to update the Control SW. If the
problem persists, please consult with your application engineer."
}
```

To solve the issue, follow the instructions in the *Message* field of the answer.

Licensing

This section provides a description of the commands related to licensing.

The T3STER SI Control Software supports several license types, but the system accepts only one *license.txt* file. In case you have multiple licenses, the licenses must to be combined into one *license.txt* file, before uploading it to the T3STER SI system.

For managing licenses, use the graphical user interface of the Control Software.

Before uploading any license to the system, the system clock must be set. For more information, see System Clock.

Query the License Status [GET_LICENSE_STATUS]

This section provides a description of the **[GET_LICENSE_STATUS]** command.

To query all active licenses and their status, use the **[GET_LICENSE_STATUS]** command:

```
{
    "Command": "GET_LICENSE_STATUS"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "HostId": "1A2B3C4D5E6F",
     "LicenseInitialized": true,
     "LicenseUpdatePresent": true,
     "Licenses": [
          {
               "Description": "Control software base license",
               "Name": "T3SIBASE",
               "Status": "OK",
               "Valid": true
          },
          {
               "Description": "Three pole measurement capability",
               "Name": "T3SI3POLE",
               "Status": "OK",
               "Valid": true
          }
     ],
     "Message": ""
}
```

The *Description* and *Name* fields contain the name and a description of the licenses that are available in the system. For more information on the available license types, see *Simcenter™ Micred™ T3STER™ SI Ultimate User Reference Guide*.

The *Valid* field indicates whether the license is valid. If the *Valid* flag is **true**, the system has a valid license of the indicated type.

The *Status* field contains a generic message. If a license is about to expire, this string will contain a warning message.

Query the Base License Status [QUERY_SYSTEM_BASE_LICENSE]

This section provides a description of the [QUERY_SYSTEM_BASE_LICENSE] command.

This is a fast query to determine the status of the base license. Use the following command:

```
{
    "Command": "QUERY_SYSTEM_BASE_LICENSE"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Chapter 3
Measurement Configurations

Configurations are the basic unit of the measurements in standalone T3STER SI systems. Configurations determine which hardware units, with what parameters, are used in a measurement.

Get a List of Existing Configurations [GET_CONFIG_LIST]

This section provides a description of the **[GET_CONFIG_LIST]** command.

To query the list of all existing configurations in the system, use the following command:

```
{
    "Command": "GET_CONFIG_LIST"
}
```

Answer message format:

```
{
    "Answer": "OK",
    "Result": [
        {
            "ConfigEditable": true/false,
            "ConfigName": Name_of_the_config,
            "Date": Date_of_last_modification_in_ISO_format,
            "Description": Description_of_the_config,
            "HwResourcesAvailable": true/false,
            "TransientAvailable": true/false,
            "TspCalibrationAvailable": true/false,
            "ConfigEditableInfo": [
generic_messages_if_config_edit_is_not_available ],
            "TransientAvailableInfo": [
generic_messages_if_transient_is_not_available ],
            "TspCalibrationAvailableInfo": [
generic_messages_if_tsp_calibration_is_not_available ]
        }
    ]
}
```

Answer message example:

```
{
    "Answer": "OK",
    "Result": [
        {
            "ConfigEditable": true,
            "ConfigName": "Diode_test",
            "Date": "2022-01-06T11:27:43Z",
            "Description": "Config for diode testing",
            "HwResourcesAvailable": true,
            "TransientAvailable": true,
            "TspCalibrationAvailable": true
        },
        {
            "ConfigEditable": true,
            "ConfigName": "MOSFET_test",
            "Date": "2021-10-04T14:58:08Z",
            "Description": "",
            "HwResourcesAvailable": true,
            "TransientAvailable": true,
            "TspCalibrationAvailable": false,
            "TspCalibrationAvailableInfo": [
                    "No tsp calibration parameters defined",
                    "No thermostat defined"
               ]
        }
    ]
}
```

The answer message contains the following mandatory fields:

- *ConfigEditable*: if the value in this field is **false**, the configuration cannot currently be modified. For example, a measurement is in progress on the configuration.
- *HwResourcesAvailable*: if the value is **false**, at least one hardware element is currently unavailable, and cannot be used for any measurement.
- *TransientAvailable*: if the value is **false**, the configuration cannot currently be used for thermal transient measurements. The possible reason is that another measurement is in progress, or something is missing from the configuration. For more information, see *Simcenter™ Micred™ T3STER™ SI Ultimate User Reference Guide*.
- *TspCalibrationAvailable*: if the value is **false**, the configuration cannot currently be used for TSP calibrations. The possible reason is that another measurement or calibration process is in progress, or something is missing from the configuration. For more information, see *Simcenter™ Micred™ T3STER™ SI Ultimate User Reference Guide*.

Optional fields:

- *TransientAvailableInfo*: if the value of the *TransientAvailable* field is **false**, then this field will contain some simple error messages: information for the user in a string array on why the transient measurement function is currently unavailable.
- *TspCalibrationAvailableInfo*: if the value of the *TspCalibrationAvailable* field is **false**, then this field will contain some simple error messages: information for the user in a string array on why the TSP calibration function is currently unavailable.
- *ConfigEditableInfo*: if the value of the *ConfigEditable* field is **false**, then this field will contain some simple error messages: information for the user in a string array on why editing the configuration is currently unavailable.

Note

The optional information fields included in the example above do not cover all possible issues.

JSON Structure of a Configuration

The JSON structure of a user defined configuration is basically (almost) the same as the HWCONFIG of the system, which can be queried by the **GET_HWCONFIG** command.

This section provides a description of the user defined configurations, but each subsection also describes the minor differences between the user defined configurations and the HWCONFIG of the system.

Configurations contain objects with **default**, **locked**, **min**, and **max** fields.

For a user configuration:

- If the **locked** field is set to **true** for a parameter, then the default value of the parameter must be used in every measurement. That is, it cannot be overwritten during measurement tasks.
- If the **locked** field is set to **false** for a parameter, then the user has the ability to change the default value of the parameter for a measurement. The value of the parameter must always be between the **min** and the **max** value.

For the HWCONFIG:

- The **min** value is the actual minimum hardware limit of the resource (except for LP220 resources).
- The **max** value is the actual maximum hardware limit of the resource (except for LP220 resources).

If a field contains a value in all capital letters, it is a special value which will be explained in subsequent sections in this document. Most of these values are special enumeration values, which are described in Enums Format.

Base JSON Structure of a Configuration

This section provides a description of the base JSON structure of a configuration.

The base of a configuration:

```
{
    "ConfigName": Name_of_the_config,
    "ConfigParams": {
        "Description": Description_of_the_config
    },
    "Resources": {
        RESOURCES
    },
    "TimingParams": {
        TIMING_PARAMS
    },
    "TspCalibParams": {
        TSP_CALIBRATION_PARAMS
    },
    "SourceTimingControl": {
        SOURCE_TIMING_PARAMS
    },
    "Type": "Config"
}
```

Variables:

- **ConfigName**:
  - The unique name of a configuration.
  - Only English alphanumeric characters, hyphens, and underscores can be used in the name of a configuration.
- **Description**:
  - The description of the configuration.
  - It is a simple text field. Everything is accepted that fits in the JSON string format.

Resources of a Configuration [Resources]

This section provides a description of the **[Resources]** in the JSON structure of a configuration.

The **[Resources]** section can contain the following fields:

```
{
    "CurrentSourceParams": CURRENT_SOURCES,
    "CurrentSourceWithActiveloadParams": CURRENT_SOURCES_WITH_ACTIVELOAD,
    "DividerParams": DIVIDERS,
    "MeasCardChParams": MEASUREMENT_CHANNELS,
    "ThermometerCardChParams": THERMOMETER_CHANNELS,
    "ThermostatParams": THERMOSTAT,
    "TriggerOutputParams": TRIGGER_OUTPUTS,
    "VoltageSourceParams": VOLTAGE_SOURCES
}
```

All fields are optional. For the HWCONFIG, the list of resource parameters depends on the currently available Hardware resources in the system.

Current Sources in a Configuration [CurrentSourceParams]

This section provides a description of the **[CurrentSourceParams]** parameters.

The structure of **[CurrentSourceParams]**:

```
{
    "CurrentSourceParams": [
        List_of_current_sources
    ]
}
```

The structure of a current source:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "SetCurrent": {
        "default": Default_current_value_in_amperes,
        "locked": true/false,
        "max": Maximum_current_value_in_amperes,
        "min": Minimum_current_value_in_amperes
    },
    "VoltageCorner": {
        "default": Default_voltage_limit_in_volts,
        "locked": true/false,
        "max": Maximum_voltage_limit_in_volts,
        "min": Minimum_voltage_limit_in_volts
    },
    "OutputMode": {
        "default": OUTPUT_MODE,
        "locked": true/false
    },
    "TriggerSource": Alias_of_the_trigger_used_for_this_source,
    "Delay": {
        "DelayFallingUs": {
            "default": Default_falling_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_falling_delay_in_microsec,
            "min": Minimum_falling_delay_in_microsec
        },
        "DelayRisingUs": {
            "default": Default_rising_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_rising_delay_in_microsec,
            "min": Minimum_rising_delay_in_microsec
        }
    }
}
```

The *TriggerSource* field is optional, and is unnecessary for all internal source elements.

The **Delay** object is optional, and is not available for all source elements.

The HWCONFIG will contain the *Delay* object if any type of delay is available for a specific source.

The *DelayRisingUs* parameter specifies the delay time interval applied after (or before, if the value is negative) the measurement trigger switches to heating state.

The *DelayFallingUs* parameter specifies the delay time interval applied after (or before, if the value is negative) the measurement trigger switches from heating to cooling state.

The HWCONFIG will contain an extra field (*DelayMode*) under the *Delay* object to describe the mode of the delay, and another optional field (*DelayModeMessage*) to provide information about the *DelayMode* on specific hardware types:

```
"DelayMode": DELAY_MODE,
    "DelayModeMessage": Text_message_about_delay_mode
```

For the actual Hardware source, if the delay only accepts discrete values, the available values are displayed in an extra field (*available*) under *DelayRisingUs*/*DelayFallingUs* in the HWCONFIG. For example.:

```
"DelayFallingUs": {
            "available": [ List_of_available_delay_values_in_microsec ],
            "default": Default_falling_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_falling_delay_in_microsec,
            "min": Minimum_falling_delay_in_microsec,
        },
```

For the actual Hardware sources, the available **OUTPUT_MODE**s are displayed in an extra field under the *OutputMode* parameter in the HWCONFIG:

```
"OutputMode": {
        "default": OUTPUT_MODE,
        "available": [ List_of_available_OUTPUT_MODEs ],
        "locked": true/false
    }
```

For special current sources (for example, MS401 current sources), there are two extra fields to describe the Hardware function more precisely:

```
"RangeLimits": [
        {
            "CurrentMax": in_amperes,
            "CurrentMin": in_amperes,
            "Symmetric": true/false,
            "VoltageMax": in_volts,
            "VoltageMin": in_volts
        }
    ],
    "VoltageCornerProgrammable": true/false
```

Variables:

- **RangeLimits**:
  - The actual current sources have multiple ranges.
  - All ranges have their own limits in current and voltage.
- **CurrentMax**: the maximum current that can be set in the specified range.
- **CurrentMin**: the minimum current that can be set in the specified range.

- **Symmetric**: if true, the range is symmetric, that is, the limits are the same with a negative sign.
- **VoltageMax**: the maximum voltage that can be set in the specified range, or the limit of the source if the value of the *VoltageCornerProgrammable* parameter is false.
- **VoltageMin**: the minimum voltage that can be set in the specified range, or the minimum voltage that should be applied on the DUT if the value of the *VoltageCornerProgrammable* parameter is false.
- **VoltageCornerProgrammable**: defines if the voltage limit can be set directly for the source or only in terms of ranges.

Current Sources with Active Load in a Configuration [CurrentSourceWithActiveloadParams]

This section provides a description of the **[CurrentSourceWithActiveloadParams]** parameters.

The structure of **[CurrentSourceWithActiveloadParams]**:

```
{
    "CurrentSourceWithActiveloadParams": [
        List_of_current_sources_with_activeload
    ]
}
```

The structure of a current source with active load:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "ActiveloadProgramming": {
        "default": ACTIVELOAD_PROGRAMMING_MODE,
        "locked": true/false
    },
    "MaxDutVoltage": {
        "default": Default_voltage_limit_in_volts,
        "locked": true/false,
        "max": Maximum_voltage_limit_in_volts,
        "min": Minimum_voltage_limit_in_volts
    },
    "OnStateDutVoltage": {
        "default": Default_voltage_limit_during_on_state_in_volts,
        "locked": true/false,
        "max": Maximum_voltage_limit_during_on_state_in_volts,
        "min": Minimum_voltage_limit_during_on_state_in_volts
    },
    "PulseLengthMs": {
        "default": Default_pulse_length_for_vc_seek_in_msec,
        "locked": true/false,
        "max": Maximum_pulse_length_for_vc_seek_in_msec,
        "min": Minimum_pulse_length_for_vc_seek_in_msec
    },
    "SetCurrent": {
        "default": Default_current_value_in_amperes,
        "locked": true/false,
        "max": Maximum_current_value_in_amperes,
        "min": Minimum_current_value_in_amperes
    },
    "OutputMode": {
        "default": OUTPUT_MODE,
        "locked": true/false
    },
    "TriggerSource": Alias_of_the_trigger_used_for_this_source,
    "Delay": {
        "DelayFallingUs": {
            "default": Default_falling_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_falling_delay_in_microsec,
            "min": Minimum_falling_delay_in_microsec
        },
        "DelayRisingUs": {
            "default": Default_rising_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_rising_delay_in_microsec,
            "min": Minimum_rising_delay_in_microsec
        }
    }}
```

The *TriggerSource* field is optional, and is unnecessary for all internal source elements.

For more information on the *Delay* object, see Current Sources in a Configuration [CurrentSourceParams]

For the actual Hardware sources, the available **OUTPUT_MODE**s are displayed in an extra field under the *OutputMode* parameter, and the available **ACTIVELOAD_PROGRAMMING_MODE**s in an extra field under the *ActiveloadProgramming* parameter in the HWCONFIG:

```
"ActiveloadProgramming": {
        "default": ACTIVELOAD_PROGRAMMING_MODE,
        "available": [ List_of_available_ACTIVELOAD_PROGRAMMING_MODEs ],
        "locked": true/false
    },
    "OutputMode": {
        "default": OUTPUT_MODE,
        "available": [ List_of_available_OUTPUT_MODEs ],
        "locked": true/false
    }
```

Dividers in a Configuration [DividerParams]

This section provides a description of the **[DividerParams]** parameters.

The structure of **[DividerParams]**:

```
{
    "DividerParams": [
        List_of_dividers
    ]
}
```

The structure of a divider:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "Range": {
        "default": Default_range_id,
        "locked": true/false,
        "max": Maximum_range_id,
        "min": Minimum_range_id
    }
}
```

The range id refers to the n^{th} element of the available division list.

For dividers, there is an extra field in the HWCONFIG, *DisplayDividerRatio*, which shows the available division rates, that is, the division list.

```
"DisplayDividerRatio": {
            "default": [
                dividers
            ],
            "locked": false
    }
```

Dividers are numerical values. If the value is 1, then there is no division during the measurement. If the number is any other numerical value (other than 1), then the measured value will be multiplied with its calibrated value.

Measurement Channels in a Configuration [MeasCardChParams]

This section provides a description of the **[MeasCardChParams]** parameters.

The structure of **[MeasCardChParams]**:

```
{
    "MeasCardChParams": [
        List_of_measurement_channels
    ]
}
```

The structure of a measurement channel:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "Sensitivity": {
        "default": [ List_of_coefficients_in_volt_per_degrees ],
        "locked": true/false,
    },
    "PowerStep": PowerStep,
    "RangeIdx": {
        "default": id,
        "locked": true/false
    },
    "AutoRange": {
        "default": true/false,
        "locked": true/false
    },
    "Uref": {
        "default": Reference_voltage_in_volts,
        "locked": true/false
    },
    "UrefSwitching": {
        "default": true/false,
        "locked": true/false
    },
    "UrefHeating": {
        "default": Reference_voltage_in_volts,
        "locked": true/false
    },
    "DividerAlias": Alias_of_the_attached_divider
}
```

Variables:

- **Sensitivity**: currently, only constant sensitivity is supported.
- **PowerStep**: for more information, see Power Step Calculations.
- **RangeIdx**: for more information, see Measurement Channels Range Definitions.
- **AutoRange**: if this parameter is set to **true**, the software defines the range automatically. A short identification pulse is applied before the start of the heating for any transient measurement, and the software tries to select the most appropriate measurement range and reference voltage (if that fails, the highest values will be selected).
- **Uref**: in the case of manual range selection, set the reference voltage setpoint for the selected range. The value represents the center of the actual measurement range.
- **UrefSwitching**: if this parameter is set to **true**, a different reference voltage (*UrefHeating*) can be set for the heating phase.
- **DividerAlias**: optional field. Used to determine the actual division rate.

Thermometer Channels in a Configuration [ThermometerCardChParams]

This section provides a description of the **[ThermometerCardChParams]** parameters.

The structure of **[ThermometerCardChParams]**:

```
{
    "ThermometerCardChParams": [
        List_of_thermometer_channels
    ]
}
```

The structure of a thermometer channel:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "Sensitivity": {
        "default": [ List_of_coefficients_in_volt_per_degrees ],
        "locked": true/false,
    },
    "PowerStep": PowerStep,
    "RangeIdx": {
        "default": id,
        "locked": true/false
    },
    "SamplePerSecIdx": {
        "default": id,
        "locked": true/false
    }
}
```

Variables:

- **Sensitivity**: this field is not currently in use. Sensitivity is calculated by the used range definition.
- **PowerStep**: for more information, see Power Step Calculations.
- **RangeIdx**: for more information, see Thermometer Channels Range Definitions.
- **SamplePerSecIdx**: optional field. For more information, see Thermometer Channels Sample Per Sec Definitions.

Thermostats in a Configuration [ThermostatParams]

This section provides a description of the **[ThermostatParams]** parameters.

The structure of **[ThermostatParams]**:

```
{
    "ThermostatParams": [
        List_of_thermostats
    ]
}
```

Note

Currently, only one thermostat can be handled by the system on the RS232 port.

The structure of a thermostat:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "SetTemperature": {
        "default": Default_value_for_set_temperature_in_degrees,
        "locked": true/false,
        "max": Maximum_value_for_set_temperature_in_degrees,
        "min": Minimum_value_for_set_temperature_in_degrees
    },
    "StabilityCriteria": {
        "DtMinMax": {
            "default":
Default_value_for_maximum_allowed_temperature_change_in_degrees,
            "locked": true/false,
            "max":
Maximum_value_for_maximum_allowed_temperature_change_in_degrees,
            "min":
Minimum_value_for_maximum_allowed_temperature_change_in_degrees
        },
        "DtTarget": {
            "default": Default_value_for_DeltaT_from_target_in_degrees,
            "locked": true/false,
            "max": Maximum_value_for_DeltaT_from_target_in_degrees,
            "min": Minimum_value_for_DeltaT_from_target_in_degrees
        },
        "TimeWindow": {
            "default": Default_value_for_time_window_in_sec,
            "locked": true/false,
            "max": Maximum_value_for_time_window_in_sec,
            "min": Minimum_value_for_time_window_in_sec
        },
        "Timeout": {
            "default": Default_value_for_timeout_in_sec,
            "locked": true/false,
            "max": Maximum_value_for_timeout_in_sec,
            "min": Minimum_value_for_timeout_in_sec
        }
    },
    "WaitForStabilityBeforeMeas": {
        "default": true/false,
        "locked": true/false
    }
}
```

Variables:

- **SetTemperature**: the temperature to be set before transient measurements.
- **StabilityCriteria**: thermostat stability parameters. For more information, see *Simcenter™ Micred™ T3STER™ SI Ultimate User Reference Guide*.
- **WaitForStabilityBeforeMeas**: if the value of this parameter is set to **true**, the system waits for thermostat stability before starting the transient measurement.

Trigger Outputs in a Configuration [TriggerOutputParams]

This section provides a description of the **[TriggerOutputParams]** parameters.

The structure of **[TriggerOutputParams]**:

```
{
    "TriggerOutputParams": [
        List_of_trigger_outputs
    ]
}
```

The structure of a trigger output:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "TriggerOutputMode": {
        "default": TRIGGER_OUTPUT_MODE,
        "locked": true/false,
    },    "Delay": {
        "DelayFallingUs": {
            "default": Default_falling_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_falling_delay_in_microsec,
            "min": Minimum_falling_delay_in_microsec
        },
        "DelayRisingUs": {
            "default": Default_rising_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_rising_delay_in_microsec,
            "min": Minimum_rising_delay_in_microsec
        }
    }}
```

For more information on the *Delay* object, see Current Sources in a Configuration [CurrentSourceParams]

For the actual Hardware sources, the available **TRIGGER_OUTPUT_MODE**s are displayed in an extra field in the HWCONFIG under the *TriggerOutputMode*:

```
"OutputMode": {
        "default": TRIGGER_OUTPUT_MODE,
        "available": [ List_of_available_TRIGGER_OUTPUT_MODEs ],
        "locked": true/false
    }
```

Voltage Sources in a Configuration [VoltageSourceParams]

This section provides a description of the **[VoltageSourceParams]** parameters.

The structure of **[VoltageSourceParams]**:

```
{
    "VoltageSourceParams": [
        List_of_voltage_sources
    ]
}
```

The structure of a voltage source:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "CurrentCorner": {
        "default": Default_current_limit_in_amperes,
        "locked": true/false,
        "max": Maximum_current_limit_in_amperes,
        "min": Minimum_current_limit__in_amperes
    },
    "OffStateVoltage": {
        "default": Default_off_state_voltage_in_volts,
        "locked": true/false,
        "max": Maximum_off_state_voltage_in_volts,
        "min": Minimum_off_state_voltage_in_volts
    },
    "OnStateVoltage": {
        "default": Default_on_state_voltage_in_volts,
        "locked": true/false,
        "max": Maximum_on_state_voltage_in_volts,
        "min": Minimum_on_state_voltage_in_volts
    },
    "ReferenceVoltage": {
        "default": Default_reference_voltage_in_volts,
        "locked": true/false,
        "max": Maximum_reference_voltage_in_volts,
        "min": Minimum_reference_voltage_in_volts
    },
    "OutputMode": {
        "default": OUTPUT_MODE,
        "locked": true/false
    },
    "TriggerSource": Alias_of_the_trigger_used_for_this_source,
    "Delay": {
        "DelayFallingUs": {
            "default": Default_falling_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_falling_delay_in_microsec,
            "min": Minimum_falling_delay_in_microsec
        },
        "DelayRisingUs": {
            "default": Default_rising_delay_in_microsec,
            "locked": true/false,
            "max": Maximum_rising_delay_in_microsec,
            "min": Minimum_rising_delay_in_microsec
        }
    }}
```

The *TriggerSource* field is optional, and is unnecessary for all internal source elements.

For more information on the *Delay* object, see Current Sources in a Configuration [CurrentSourceParams]

For the actual Hardware sources, the available **OUTPUT_MODE**s are displayed in an extra field in the HWCONFIG under the *OutputMode*:

```
"OutputMode": {
        "default": OUTPUT_MODE,
        "available": [ List_of_available_OUTPUT_MODEs ],
        "locked": true/false
    }
```

Timing Parameters in a Configuration [TimingParams]

This section provides a description of the **[TimingParams]** parameters.

The structure of **[TimingParams]**:

```
"TimingParams": {
        "CoolingTime": {
            "default": Default_cooling_time_in_sec,
            "locked": false,
            "max": Maximum_cooling_time_in_sec,
            "min": Minimum_cooling_time_in_sec
        },
        "DelayTime": {
            "default": Default_delay_time_in_sec,
            "locked": false,
            "max": Maximum_delay_time_in_sec,
            "min": Minimum_delay_time_in_sec
        },
        "HeatingTime": {
            "default": Default_heating_time_in_sec,
            "locked": false,
            "max": Maximum_heating_time_in_sec,
            "min": Minimum_heating_time_in_sec
        },
        "TransientMode": {
            "default": "Cooling",
            "locked": false
        },
        "SamplePerOctave": {
            "default": Default_sample_per_octave,
            "locked": false,
            "max": Maximum_sample_per_octave,
            "min": Minimum_sample_per_octave
        },
        "Repeat": {
            "default": Default_repeats,
            "locked": false,
            "max": Maximum_repeats,
            "min": Minimum_repeats
        }
    }
```

Variables:

- **TransientMode**: currently, only the **"Cooling"** option is supported. This will make both the heating and cooling measurements.
- **SamplePerOctave**: currently, only the 1000 sample-per-octave value is supported.
- **Repeat**: the **Repeat** option is currently not supported. Use the value **1**.

TSP Calibration Parameters in a Configuration [TspCalibParams]

This section provides a description of the **[TspCalibParams]** parameters.

The structure of **[TspCalibParams]**:

```
"TspCalibParams": {
        "CustomTemperature": {
            "default":
Default_custom_temperature_for_endaction_in_degrees,
            "locked": true/false,
            "max": Maximum_custom_temperature_for_endaction_in_degrees,
            "min": Minimum_custom_temperature_for_endaction_in_degrees
        },
        "DutStability": {
            "default": true/false,
            "locked": true/false,
        },
        "EndAction": {
            "default": END_ACTION,
            "locked": true/false,
        },
        "Mode": {
            "default": CALIBRATION_DIRECTION,
            "locked": true/false,
        },
        "ThtIntSensor": {
            "default": true/false,
            "locked": true/false,
        },
        "Tmax": {
            "default":
Default_of_maximum_temperature_for_calibration_in_degrees,
            "locked": true/false,
            "max":
Maximum_of_maximum_temperature_for_calibration_in_degrees,
            "min":
Minimum_of_maximum_temperature_for_calibration_in_degrees
        },
        "Tmin": {
            "default":
Default_of_minimum_temperature_for_calibration_in_degrees,
            "locked": true/false,
            "max":
Maximum_of_minimum_temperature_for_calibration_in_degrees,
            "min":
Minimum_of_minimum_temperature_for_calibration_in_degrees
        },
        "Tstep": {
            "default":
Default_of_temperature_step_for_calibration_in_degrees,
            "locked": true/false,
      "max": Maximum_of_temperature_step_for_calibration_in_degrees,
"min": Minimum_of_temperature_step_for_calibration_in_degrees
        }
    }
```

Variables:

- **DutStability**: if the value is set to **true**, the system waits for DUT stability before starting to record the calibration data.
- **ThtIntSensor**: if the value is set to **true**, the system waits for thermostat stability before starting to record the calibration data.

Source Timing Control Parameters in a Configuration [SourceTimingControl]

This section provides a description of the **[SourceTimingControl]** parameters.

The structure of **[SourceTimingControl]**:

```
"SourceTimingControl": {
        "Enabled" : true,
        "ReversePowerOff" : true,
        "locked" : false,
        "WaitForInstrumentDelay" : true,
        "PowerOn" : [
            List_of_source_timing_control_steps
        ],
        "PowerOff" : [
            List_of_source_timing_control_steps
        ]
    }
```

[SourceTimingControl] is an optional structure. If omitted, a default power on/off will be performed, with appropiate instrument delays.

Variables:

- **ReversePowerOff**: if the value is set to **true**, the *PowerOff* sequence will be a reversed *PowerOn* sequence.
- **WaitForInstrumentDelay**: if the value is set to **true**, the Control Software will wait out instrument delays before starting a measurement or calibration in case not enough sleep time was specified in the *SleepAfter* field in *PowerOn*.

The structure of a source timing control step:

```
{
    "Alias": Unique_system_selected_alias,
    "Type": SOURCE_TIMING_CONTROL_RESOURCE_TYPE,
    "SleepAfter": Sleep_in_sec_after_the_step
}
```

Variables:

- **Alias**: unique system selected alias of the resource.

- **SleepAfter**: sleep time in seconds after the source is powered on/off. At least the time specified in this field will pass before starting the next step. The maximum value is 10sec.

Get the Actual Hardware Configuration [GET_HWCONFIG]

This section provides a description of the **[GET_HWCONFIG]** command.

Use this command to identify all available Hardware elements of the actual system and their parameters.

To query the list of all existing Hardware elements in the system, use the following command:

```
{
    "Command": "GET_HWCONFIG"
}
```

Answer message format:

```
{
    "ConfigName": "DEFAULT",
    "ConfigParams": {
        "Description": ""
    },
    "Resources": {
        RESOURCES
    },
    "TimingParams": {
        TIMING_PARAMS
    },
    "TspCalibParams": {
        TSP_CALIBRATION_PARAMS
    },
    "SourceTimingControl": {
        SOURCE_TIMING_PARAMS
    }}
```

For more information on the elements of the answer message, see Base JSON Structure of a Configuration.

Query an Existing Measurement Configuration [GET_CONFIG]

This section provides a description of the **[GET_CONFIG]** command.

To query an already saved configuration, use the following command:

```
{
    "Command": "GET_CONFIG",
    "ConfigName": Name_of_the_config
}
```

Command example:

```
{
    "Command": "GET_CONFIG",
    "ConfigName": "MyTestConfig"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "ConfigEditable": true/false,
     "ConfigName": Name_of_the_config,
     "ConfigParams": {
          "Description": Description_of_the_config
     },
     "HwResourcesAvailable": true/false,
     "Resources": {
        RESOURCES
     },
     "TimingParams": {
        TIMING_PARAMS
    },
     "TransientAvailable": true/false,
     "TspCalibrationAvailable": true/false,
     "Type": "Config"
}
```

For more information on the elements of the answer message, see Base JSON Structure of a Configuration.

For more information on the fields related to availability (for example, *HwResourcesAvailable*), see Get a List of Existing Configurations [GET_CONFIG_LIST].

Remove an Existing Measuerement Configuration [REMOVE_CONFIG]

This section provides a description of the **[REMOVE_CONFIG]** command.

To remove a saved configuration, use the following command:

```
{
    "Command": "REMOVE_CONFIG",
    "ConfigName": Name_of_the_config
}
```

Command example:

```
{
    "Command": "REMOVE_CONFIG",
    "ConfigName": "MyTestConfig"
}
```

If the remove operation is successful, the answer is **"OK"**:

```
{
    "Answer": "OK"
}
```

Save a Measurement Configuration [SAVE_CONFIG]

This section provides a description of the **[SAVE_CONFIG]** command.

To save a configuration, use the following command:

```
{
    "Command": "SAVE_CONFIG",
    "Type": "Config",
    "ConfigName": Name_of_the_config,
    "ConfigParams": {
        "Description": Description_of_the_config
    },
    "Resources": {
        RESOURCES
    },
    "TimingParams": {
        TIMING_PARAMS
    },
    "TspCalibParams": {
        TSP_CALIBRATION_PARAMS
    },
    "SourceTimingControl": {
        SOURCE_TIMING_PARAMS
    }}
```

The *TspCalibParams* and *SourceTimingControl* fields are optional.

For more information on the elements of the answer message, see Base JSON Structure of a Configuration.

Command example:


```
{
    "Command": "SAVE_CONFIG",
    "Type": "Config",
    "ConfigName": "MyTestConfig",
    "ConfigParams": {
        "Description": "This is the description of my config."
    },
    "Resources": {
        "CurrentSourceParams": [
            {
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "",
                "OutputMode": {
                    "default": "ON",
                    "locked": false
                },
                "SetCurrent": {
                    "min": -0.2,
                    "max": 0.2,
                    "default": 0.01,
                    "locked": false
                },
                "VoltageCorner": {
                    "min": -40,
                    "max": 40,
                    "default": 10,
                    "locked": false
                }
            },
            {
                "Alias": "/T3STER/0/LP220/SLOT1/CH0",
                "UserAlias": "",
                "OutputMode": {
                    "default": "PC",
                    "locked": false
                },
                "SetCurrent": {
                    "min": -2,
                    "max": 2,
                    "default": 0.1,
                    "locked": false
                },
                "VoltageCorner": {
                    "min": -10,
                    "max": 10,
                    "default": 2,
                    "locked": true
                }
            }
        ],
        "MeasCardChParams": [
            {
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "",
                "Sensitivity": {
```

```
                    "default": [
                        0.002
                    ],
                    "locked": false
                },
               "PowerStep": "@POWERSTEP_DIODE;/T3STER/0/MS401/SLOT3/CH0;/
T3STER/0/LP220/SLOT1/CH0",
                "AutoRange": {
                    "default": false,
                    "locked": false
                },
                "RangeIdx": {
                    "default": 13,
                    "locked": false
                },
                "Uref": {
                    "default": 0.5,
                    "locked": false
                }
            }
        ]
    },
    "TimingParams": {
        "TransientMode": {
            "default": "Cooling",
            "locked": false
        },
        "HeatingTime": {
            "min": 0,
            "max": 4000,
            "default": 15,
            "locked": false
        },
        "CoolingTime": {
            "min": 0,
            "max": 4000,
            "default": 15,
            "locked": false
        },
        "DelayTime": {
            "min": 0,
            "max": 4000,
            "default": 0,
            "locked": false
        },
        "SamplePerOctave": {
            "default": 1000,
            "locked": false,
            "max": 1000,
            "min": 10
        },
        "Repeat": {
            "default": 1,
            "locked": false,
            "max": 100,
            "min": 1
        }
    }
```

```
}
```

Answer message:

```
{
      "Answer": "OK"
}
```

Chapter 4
Tasks (Measurements)

Tasks are used to manage measurements.

There are four types of measurement tasks:

- Monitoring (*TaskMode*: **MONITORING**)
- TSP calibration (*TaskMode*: **TSPCALIB**)
- Identification (*TaskMode*: **IDENTIFICATION**)
- Thermal transient measurement (*TaskMode*: **TRANSIENT**)

  A task can only be started on an existing and currently available configuration. Before starting a task, you have to allocate the resources by initiating a **MONITORING_RESOURCE_ALLOCATION** task.

  Note

  Only one resource allocation is allowed at a time and only one measurement task is allowed at a time for one existing resource allocation.

  Each task must have a *TaskAlias*, defined in the **START_TASK** command. The task can later be queried or managed based on this parameter.

JSON Structure of a Measurement Model

Measurement models are used in measurement commands, and in model queries.

A measurement model is very similar to a measurement configuration. There are two types of measurement models:

- **TransientModel**
- **TspCalibrationModel**

Base of the TransientModel

The **TransientModel** is used in commands related to transient measurements.

The base of the **TransientModel**:

```
{
    "ConfigName": Name_of_the_config,
    "Resources": {
        RESOURCES
    },
    "TimingParams": {
        TIMING_PARAMS
    },
    "SourceTimingControl": {
        SOURCE_TIMING_PARAMS
    },
    "Type": "TransientModel"
}
```

Base of the TspCalibrationModel

The **TspCalibrationModel** is used in commands related to TSP calibrations.

The base of the **TspCalibrationModel**:

```
{
    "ConfigName": Name_of_the_config,
    "Resources": {
        RESOURCES
    },
    "TspCalibParams": {
        TSP_CALIBRATION_PARAMS
    },
    "SourceTimingControl": {
        SOURCE_TIMING_PARAMS
    },
    "Type": "TspCalibrationModel"
}
```

Resources of Measurement Models [Resources]

This section provides a description of the **[Resources]** in a measurement model.

The **[Resources]** section can contain the following fields (same as in measurement configurations):

```
{
    "CurrentSourceParams": CURRENT_SOURCES,
    "CurrentSourceWithActiveloadParams": CURRENT_SOURCES_WITH_ACTIVELOAD,
    "DividerParams": DIVIDERS,
    "MeasCardChParams": MEASUREMENT_CHANNELS,
    "ThermometerCardChParams": THERMOMETER_CHANNELS,
    "ThermostatParams": THERMOSTAT,
    "TriggerOutputParams": TRIGGER_OUTPUTS,
    "VoltageSourceParams": VOLTAGE_SOURCES
}
```

You have to use the same resources that are specified in the measurement configuration. It is not allowed to use a resource that is not specified in the configuration. In case a resource that is specified in the configuration is not included in the measurement model, the software programs that resource to its default value.

Current Sources in Measurement Models [CurrentSourceParams]

This section provides a description of the **[CurrentSourceParams]** parameters in measurement models.

The structure of **[CurrentSourceParams]**:

```
{
    "CurrentSourceParams": [
        List_of_current_sources
    ]
}
```

The structure of a current source:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "SetCurrent": Current_value_in_amperes,
    "VoltageCorner": Voltage_limit_in_volts,
    "OutputMode": OUTPUT_MODE,
    "TriggerSource": Alias_of_the_trigger_used_for_this_source,
    "Delay": {
        "DelayFallingUs": Falling_delay_in_microsec,
        "DelayRisingUs": Rising_delay_in_microsec
    }}
```

The *TriggerSource* field is optional, and is unnecessary for all internal source elements.

The *Delay* field is optional. For more information on the *Delay* object, see Current Sources in a Configuration [CurrentSourceParams]

Current Sources with Active Load in Measurement Models [CurrentSourceWithActiveloadParams]

This section provides a description of the **[CurrentSourceWithActiveloadParams]** parameters in measurement models.

The structure of **[CurrentSourceWithActiveloadParams]**:

```
{
    "CurrentSourceWithActiveloadParams": [
        List_of_current_sources_with_activeload
    ]
}
```

The structure of a current source with active load:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "ActiveloadProgramming": ACTIVELOAD_PROGRAMMING_MODE,
    "MaxDutVoltage": Voltage_limit_in_volts,
    "OnStateDutVoltage": Voltage_limit_during_on_state_in_volts,
    "PulseLengthMs": Pulse_length_for_vc_seek_in_msec,
    "SetCurrent": Current_value_in_amperes,
    "OutputMode": OUTPUT_MODE,
    "TriggerSource": Alias_of_the_trigger_used_for_this_source,
    "Delay": {
        "DelayFallingUs": Falling_delay_in_microsec,
        "DelayRisingUs": Rising_delay_in_microsec
    }}
```

The *TriggerSource* field is optional, and is unnecessary for all internal source elements.

The *Delay* field is optional. For more information on the *Delay* object, see Current Sources in a Configuration [CurrentSourceParams]

Dividers in Measurement Models [DividerParams]

This section provides a description of the **[DividerParams]** parameters in measurement models.

The structure of **[DividerParams]**:

```
{
    "DividerParams": [
        List_of_dividers
    ]
}
```

The structure of a divider:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "Range": Range_id
}
```

The range id refers to the n^{th} element of the available division list.

Measurement Channels in Measurement Models [MeasCardChParams]

This section provides a description of the **[MeasCardChParams]** parameters in measurement models.

The structure of **[MeasCardChParams]**:

```
{
    "MeasCardChParams": [
        List_of_measurement_channels
    ]
}
```

The structure of a measurement channel:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "Sensitivity": [ List_of_coefficients_in_volt_per_degrees ],
    "PowerStep": PowerStep,
    "RangeIdx": id,
    "AutoRange": true/false,
    "Uref": Reference_voltage_in_volts,
    "UrefSwitching": true/false,
    "UrefHeating": Reference_voltage_in_volts,
    "DividerAlias": Alias_of_the_attached_divider
}
```

Variables:

- **Sensitivity**: currently, only constant sensitivity is supported.
- **PowerStep**: for more information, see Power Step Calculations.
- **RangeIdx**: for more information, see Measurement Channels Range Definitions.
- **AutoRange**: if this parameter is set to **true**, the software defines the range automatically. A short identification pulse is applied before the start of the heating for any transient measurement, and the software tries to select the most appropriate measurement range and reference voltage.
- **Uref**: in the case of manual range selection, set the reference voltage setpoint for the selected range. The value represents the center of the actual measurement range.
- **UrefSwitching**: if this parameter is set to **true**, a different reference voltage (*UrefHeating*) can be set for the heating phase.
- **DividerAlias**: optional field. Used to determine the actual division rate.

Thermometer Channels in Measurement Models [ThermometerCardChParams]

This section provides a description of the **[ThermometerCardChParams]** parameters in measurement models.

The structure of **[ThermometerCardChParams]**:

```
{
    "ThermometerCardChParams": [
        List_of_thermometer_channels
    ]
}
```

The structure of a thermometer channel:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "Sensitivity": [ List_of_coefficients_in_volt_per_degrees ],
    "PowerStep": PowerStep,
    "RangeIdx": id,
    "SamplePerSecIdx": id
}
```

Variables:

- **Sensitivity**: this field is not currently in use. Sensitivity is calculated by the used range definition.
- **PowerStep**: for more informati.on, see Power Step Calculations.
- **RangeIdx**: for more information, see Thermometer Channels Range Definitions.
- **SamplePerSecIdx**: optional field. For more information, see Thermometer Channels Sample Per Sec Definitions.

Thermostats in Measurement Models [ThermostatParams]

This section provides a description of the **[ThermostatParams]** parameters in measurement models.

The structure of **[ThermostatParams]**:

```
{
    "ThermostatParams": [
        List_of_thermostats
    ]
}
```

Note

Currently, only one thermostat can be handled by the system on the RS232 port.

The structure of a thermostat:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "SetTemperature": Set_temperature_in_degrees,
    "StabilityCriteria": {
        "DtMinMax": Maximum_allowed_temperature_change_in_degrees,
        "DtTarget": Allowed_DeltaT_from_target_in_degrees,
        "TimeWindow": Time_window_in_sec,
        "Timeout": Timeout_in_sec
    },
    "WaitForStabilityBeforeMeas": true/false
}
```

Variables:

- **SetTemperature**: the temperature to be set before transient measurements.
- **StabilityCriteria**: thermostat stability parameters. For more information, see *Simcenter™ Micred™ T3STER™ SI Ultimate User Reference Guide*.
- **WaitForStabilityBeforeMeas**: if the value of this parameter is set to **true**, the system waits for thermostat stability before starting the transient measurement.

Trigger Outputs in Measurement Models [TriggerOutputParams]

This section provides a description of the **[TriggerOutputParams]** parameters in measurement models.

The structure of **[TriggerOutputParams]**:

```
{
    "TriggerOutputParams": [
        List_of_trigger_outputs
    ]
}
```

The structure of a trigger output:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
   "TriggerOutputMode": TRIGGER_OUTPUT_MODE,
    "Delay": {
        "DelayFallingUs": Falling_delay_in_microsec,
        "DelayRisingUs": Rising_delay_in_microsec
    }}
```

The *Delay* field is optional. For more information on the *Delay* object, see Current Sources in a Configuration [CurrentSourceParams]

Voltage Sources in Measurement Models [VoltageSourceParams]

This section provides a description of the **[VoltageSourceParams]** parameters in measurement models.

The structure of **[VoltageSourceParams]**:

```
{
    "VoltageSourceParams": [
        List_of_voltage_sources
    ]
}
```

The structure of a voltage source:

```
{
    "Alias": Unique_system_selected_alias,
    "UserAlias": User_defined_alias,
    "CurrentCorner": Current_limit_in_amperes,
    "OffStateVoltage": Off_state_voltage_in_volts,
    "OnStateVoltage": On_state_voltage_in_volts,
    "ReferenceVoltage": Reference_voltage_in_volts,
    "OutputMode": OUTPUT_MODE,
    "TriggerOutputMode": TRIGGER_OUTPUT_MODE,
    "Delay": {
        "DelayFallingUs": Falling_delay_in_microsec,
        "DelayRisingUs": Rising_delay_in_microsec
    }}
```

The *TriggerSource* field is optional, and is unnecessary for all internal source elements.

The *Delay* field is optional. For more information on the *Delay* object, see Current Sources in a Configuration [CurrentSourceParams]

Timing Parameters in Measurement Models [TimingParams]

This section provides a description of the **[TimingParams]** parameters in measurement models.

The structure of **[TimingParams]**:

```
"TimingParams": {
        "CoolingTime": Cooling_time_in_sec,
        "DelayTime": Delay_time_in_sec,
        "HeatingTime": Heating_time_in_sec,
        "TransientMode": "Cooling",
        "SamplePerOctave": Sample_per_octave,
        "Repeat": Repeats
    }
```

Variables:

- **TransientMode**: currently, only the **"Cooling"** option is supported. This will make both the heating and cooling measurements.
- **SamplePerOctave**: currently, only the 1000 sample-per-octave value is supported.
- **Repeat**: the **Repeat** option is currently not supported. Use the value **1**.

TSP Calibration Parameters in Measurement Models [TspCalibParams]

This section provides a description of the **[TspCalibParams]** parameters in measurement models.

The structure of **[TspCalibParams]**:

```
"TspCalibParams": {
        "CustomTemperature": Custom_temperature_for_endaction_in_degrees,
        "DutStability": true/false,
        "EndAction": END_ACTION,
        "Mode": CALIBRATION_DIRECTION,
        "ThtIntSensor": true/false,
        "Tmax": Maximum_temperature_for_calibration_in_degrees,
        "Tmin": Minimum_temperature_for_calibration_in_degrees,
        "Tstep": Temperature_step_for_calibration_in_degrees
    }
```

Variables:

- **DutStability**: if the value is set to **true**, the system waits for DUT stability before starting to record the calibration data.
- **ThtIntSensor**: if the value is set to **true**, the system waits for thermostat stability before starting to record the calibration data.

Source Timing Control Parameters in Measurement Models [SourceTimingControl]

This section provides a description of the **[SourceTimingControl]** parameters in measurement models.

The structure of **[SourceTimingControl]**:

```
"SourceTimingControl": {
        "Enabled" : true,
        "ReversePowerOff" : true,
        "WaitForInstrumentDelay" : true,
        "PowerOn" : [
            List_of_source_timing_control_steps
        ],
        "PowerOff" : [
            List_of_source_timing_control_steps
        ]
    }
```

[SourceTimingControl] is an optional structure. If omitted, a default power on/off will be performed, with appropiate instrument delays.

Variables:

- **ReversePowerOff**: if the value is set to **true**, the *PowerOff* sequence will be a reversed *PowerOn* sequence.
- **WaitForInstrumentDelay**: if the value is set to **true**, the Control Software will wait out instrument delays before starting a measurement or calibration in case not enough sleep time was specified in the *SleepAfter* field in *PowerOn*.

The structure of a source timing control step:

```
{
    "Alias": Unique_system_selected_alias,
    "Type": SOURCE_TIMING_CONTROL_RESOURCE_TYPE,
    "SleepAfter": Sleep_in_sec_after_the_step
}
```

Variables:

- **Alias**: unique system selected alias of the resource.
- **SleepAfter**: sleep time in seconds after the source is powered on/off. At least the time specified in this field will pass before starting the next step or the measurement. The maximum value is 10sec.

Allocate Resources for Measurements [MONITORING_RESOURCE_ALLOCATION]

Resource allocation is required for all measurements.

Note

The Control Software starts the resource allocation automatically when you enter the **Calibration** or **Measurement** view.

When using the API, you have to initiate resource allocation manually (using the **[START_TASK - MONITORING_RESOURCE_ALLOCATION]** command.

Start Allocation [START_TASK - MONITORING_RESOURCE_ALLOCATION]

This section provides a description of the [START_TASK - MONITORING_RESOURCE_ALLOCATION] command.

Command format:

```
{
    "Command": "START_TASK",
    "TaskMode": "MONITORING_RESOURCE_ALLOCATION",
    "ConfigName": Name_of_the_config,
    "TaskAlias": Name_of_the_task,
    "LoadConfig": true
}
```

Note

The value of the *TaskAlias* parameter can only contain English alphanumeric characters, hyphens, and underscores.

Note

The Control Software uses the configuration name for the *TaskAlias* parameter in resource allocation.

Note

There is an optional parameter (*HandleUserDisconnect*: false/true), which determines the backend's behavior on disconnecting from the websocket.

The default behavior is that if the user disconnects from the websocket, the resource allocation and monitoring tasks will be stopped. Measurement tasks, like thermal transients, are not affected by this behavior: they will continue to run in case the user disconnects.
This behavior can be disabled by *HandleUserDisconnect*: false. It is recommended to maintain websocket connection (and not use this optional parameter).

Command example:

```
{
    "Command": "START_TASK",
    "TaskMode": "MONITORING_RESOURCE_ALLOCATION",
    "ConfigName": "MyTestConfig",
    "TaskAlias": "MyTestConfig",
    "LoadConfig": true
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Answer message example if resource allocation cannot be started, because, for example, another allocation procedure is in progress:

```
{
     "Answer": "CANNOT_START",
     "Message": "INSUFFICIENT_RESOURCES"
}
```

End Allocation [STOP_AND_REMOVE_TASK]

This section provides a description of the **[STOP_AND_REMOVE_TASK]** command.

Command format:

```
{
    "Command": "STOP_AND_REMOVE_TASK",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "STOP_AND_REMOVE_TASK",
    "TaskAlias": "MyTestConfig"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Monitoring Tasks [MONITORING]

Before starting a monitoring task for a resource, you must initiate a resource allocation.

The monitoring task will update the resource allocation task, and turn on the sources with their cooling state values. If you want to turn off the sources and go back to the resource allocation state, you need to update the task back to resource allocation.

Note

In case you are using the Control Software, the monitoring task is started automatically when you press the **Enable Sources** button. The monitoring task stops when you press the **Disable Sources** button.

Start a Monitoring Task [UPDATE_TASK - MONITORING]

This section provides a description of the **[UPDATE_TASK - MONITORING]** command.

Command format:

```
{
    "Command": "UPDATE_TASK",
    "TaskMode": "MONITORING",
    "ConfigName": Name_of_the_config,
    "TaskAlias": Name_of_the_task,
    "Type": "TransientModel",
    "Resources": {
        RESOURCES
    }
}
```

Note

The value of the *TaskAlias* parameter must be the same as in the **[START_TASK - MONITORING_RESOURCE_ALLOCATION]** command for the same resource allocation.

Command example:


```
{
    "Command": "UPDATE_TASK",
    "TaskMode": "MONITORING",
    "ConfigName": "MyTestConfig",
    "TaskAlias": "MyTestConfig",
    "Type": "TransientModel",
    "Resources": {
        "CurrentSourceParams": [{
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "",
                "OutputMode": "ON",
                "SetCurrent": 0.001,
                "VoltageCorner": 10
            }, {
                "Alias": "/T3STER/0/LP220/SLOT1/CH0",
                "UserAlias": "",
                "OutputMode": "PC",
                "SetCurrent": 0.1,
                "VoltageCorner": 2
            }
        ],
        "MeasCardChParams": [{
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "",
                "Sensitivity": [0.002],
               "PowerStep": "@POWERSTEP_DIODE;/T3STER/0/MS401/SLOT3/CH0;/
T3STER/0/LP220/SLOT1/CH0",
                "AutoRange": false,
                "RangeIdx": 13,
                "Uref": 0.5
            }
        ]
    }
}
```


Answer message example:


```
{
     "Answer": "OK",
     "Message": ""
}
```


Stop a Monitoring Task [UPDATE_TASK - MONITORING_RESOURCE_ALLOCATION]

This section provides a description of the **[UPDATE_TASK - MONITORING_RESOURCE_ALLOCATION]** command.

Command format:

```
{
    "Command":"UPDATE_TASK",
    "TaskMode":"MONITORING_RESOURCE_ALLOCATION",
    "ConfigName": Name_of_the_config,
    "TaskAlias": Name_of_the_task
}
```

Note

The value of the *TaskAlias* parameter must be the same as in the **[START_TASK - MONITORING_RESOURCE_ALLOCATION]** command for the same resource allocation.

Command example:

```
{
    "Command":"UPDATE_TASK",
    "TaskMode":"MONITORING_RESOURCE_ALLOCATION",
    "ConfigName":"MyTestConfig",
    "TaskAlias":"MyTestConfig"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

TSP Calibration [TSPCALIB]

Before starting a TSP calibration task, you must initiate a resource allocation. The calibration task will run alongside the resource allocation task.

A TSP calibration task can be stopped any time during the calibration, but in this case no results will be returned.

When a TSP calibration task is finished, the results can be downloaded from the T3STER SI. For more information, see Query the Links to the Result Files [QUERY_TASK_RESULT_FILE_LIST].

Before starting a new task, you must remove any finished task using the **[REMOVE_TASK - TSPCALIB]** command.

Start a TSP Calibration [START_TASK - TSPCALIB]

This section provides a description of the **[START_TASK - TSPCALIB]** command.

Command format:

```
{
    "Command": "START_TASK",
    "TaskMode": "TSPCALIB",
    "ConfigName": Name_of_the_config,
    "TaskAlias": Name_of_the_task,
    "Type": "TspCalibrationModel",
    "Resources": {
        RESOURCES
    },
    "TspCalibParams": {
        TSP_CALIBRATION_PARAMS
    },
    "SourceTimingControl": {
        SOURCE_TIMING_PARAMS
    }
}
```

Note

The value of the *TaskAlias* parameter can only contain English alphanumeric characters, hyphens, and underscores.

Note

The Control Software uses the configuration name, with the **_calibration** string, for the *TaskAlias* parameter in TSP calibration tasks. For example, **"MyTestConfig_calibration"**.

Note

SourceTimingControl is an optional parameter. If omitted, a default power on/off will be performed, with appropriate instrument delays.

Command example:

```
{
    "Command": "START_TASK",
    "TaskMode": "TSPCALIB",
    "ConfigName": "MyTestConfig",
    "TaskAlias": "MyTestConfig_calibration",
    "Type": "TspCalibrationModel",
    "Resources":{
        "CurrentSourceParams":[
            {
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "S3Ch1",
                "OutputMode": "ON",
                "SetCurrent": 0.01,
                "VoltageCorner": 10
            },
            {
                "Alias": "/T3STER/0/LP220/SLOT1/CH0",
                "UserAlias": "S1Ch1",
                "OutputMode": "PC",
                "SetCurrent": 0.1,
                "VoltageCorner": 2
            }
        ],
        "MeasCardChParams":[
            {
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "S3Ch1",
               "PowerStep": "@POWERSTEP_DIODE;/T3STER/0/MS401/SLOT3/CH0;/
T3STER/0/LP220/SLOT1/CH0",
                "AutoRange": false,
                "RangeIdx": 13,
                "Uref": 0.5,
                "UrefSwitching": false,
                "UrefHeating": 0
            }
        ],
        "ThermostatParams":[
            {
                "Alias": "/THERMOSTAT/0",
                "UserAlias": "Th0",
                "SetTemperature": 25,
                "StabilityCriteria": {
                    "TimeWindow": 15,
                    "DtMinMax": 1,
                    "DtTarget": 2,
                    "Timeout": 600
                },
                "WaitForStabilityBeforeMeas": false
            }
        ]
    },
    "TspCalibParams": {
        "Tmin": 25,
        "Tmax": 85,
        "Tstep": 10,
```

```
            "Mode": "Upwards",
            "ThtIntSensor": true,
            "DutStability": false,
            "EndAction": "StartTemp",
            "CustomTemperature": 25
        }
    }
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Stop a TSP Calibration [STOP_TASK - TSPCALIB]

This section provides a description of the **[STOP_TASK - TSPCALIB]** command.

Command format:

```
{
    "Command": "STOP_TASK",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "STOP_TASK",
    "TaskAlias": "MyTestConfig_calibration"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Remove a TSP Calibration Task [REMOVE_TASK - TSPCALIB]

This section provides a description of the **[REMOVE_TASK - TSPCALIB]** command.

A task must be stopped or finished before it can be removed.

Command format:

```
{
    "Command": "REMOVE_TASK",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "REMOVE_TASK",
    "TaskAlias": "MyTestConfig_calibration"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Identification Tasks [IDENTIFICATION]

Before starting an identification task, you must initiate a resource allocation. The identification task will run alongside the resource allocation task.

An identification task can be stopped any time during the identification procedure, but in this case no results will be returned.

When an identification task is finished, the identified parameters can be queried with the transient model **[QUERY_TASK_TRANSIENTMODEL]**.

Before starting a new task, you must remove any finished task using the **[REMOVE_TASK - IDENTIFICATION]** command.

Start an Identification Task [START_TASK - IDENTIFICATION]

This section provides a description of the **[START_TASK - IDENTIFICATION]** command.

Command format:

```
{
    "Command": "START_TASK",
    "TaskMode": "IDENTIFICATION",
    "ConfigName": Name_of_the_config,
    "TaskAlias": Name_of_the_task,
    "Type": "TransientModel",
    "Resources": {
        RESOURCES
    },
    "TimingParams": {
        TIMING_PARAMS
    },
    "SourceTimingControl": {
        SOURCE_TIMING_PARAMS
    }
}
```

Note

The value of the *TaskAlias* parameter can only contain English alphanumeric characters, hyphens, and underscores.

Note

The Control Software uses the configuration name, with the **_identification** string, for the *TaskAlias* parameter in identification tasks. For example, **"MyTestConfig_identification"**.

Note

SourceTimingControl is an optional parameter. If omitted, a default power on/off will be performed, with appropriate instrument delays.

Command example:


```
{
    "Command": "START_TASK",
    "TaskMode": "IDENTIFICATION",
    "ConfigName": "MyTestConfig",
   "TaskAlias": "MyTestConfig_identification",
    "Type": "TransientModel",
    "Resources": {
        "CurrentSourceParams": [{
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "S3Ch1",
                "OutputMode": "ON",
                "SetCurrent": 0.01,
                "VoltageCorner": 10
            }, {
                "Alias": "/T3STER/0/LP220/SLOT1/CH0",
                "UserAlias": "S1Ch1",
                "OutputMode": "PC",
                "SetCurrent": 0.1,
                "VoltageCorner": 2
            }
        ],
        "MeasCardChParams": [{
                "Sensitivity": [
                    0.002
                ],
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "S3Ch1",
               "PowerStep": "@POWERSTEP_DIODE;/T3STER/0/MS401/SLOT3/CH0;/
T3STER/0/LP220/SLOT1/CH0",
                "AutoRange": true,
                "RangeIdx": 13,
                "Uref": 0.5,
                "UrefSwitching": false,
                "UrefHeating": 0,
                "DividerAlias": ""
            }
        ]
    },
   "TimingParams":{
        "TransientMode":"Cooling",
        "HeatingTime":15,
        "CoolingTime":15,
        "DelayTime":0,
        "SamplePerOctave":1000,
        "Repeat":1
    }
}
```


Answer message example:


```
{
     "Answer": "OK",
     "Message": ""
}
```

Stop an Identification Task [STOP_TASK - IDENTIFICATION]

This section provides a description of the **[STOP_TASK - IDENTIFICATION]** command.

Command format:

```
{
    "Command": "STOP_TASK",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "STOP_TASK",
    "TaskAlias": "MyTestConfig_identification"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Remove an Identification Task [REMOVE_TASK - IDENTIFICATION]

This section provides a description of the **[REMOVE_TASK - IDENTIFICATION]** command.

A task must be stopped or finished before it can be removed.

Command format:

```
{
    "Command": "REMOVE_TASK",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "REMOVE_TASK",
    "TaskAlias": "MyTestConfig_identification"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Thermal Transient Measurement Tasks [TRANSIENT]

Before starting a thermal transient measurement task, you must initiate a resource allocation. The measurement task will run alongside the resource allocation task.

A thermal transient measurement task can be stopped any time during the measurement, but in this case no results will be returned.

When a thermal transient measurement task is finished, the results can be downloaded from the T3STER SI. For more information, see Query the Links to the Result Files [QUERY_TASK_RESULT_FILE_LIST].

Before starting a new task, you must remove any finished task using the **[REMOVE_TASK - TRANSIENT]** command.

Start a Thermal Transient Measurement Task [START_TASK - TRANSIENT]

This section provides a description of the **[START_TASK - TRANSIENT]** command.

Command format:

```
{
    "Command": "START_TASK",
    "TaskMode": "TRANSIENT",
    "ConfigName": Name_of_the_config,
    "TaskAlias": Name_of_the_task,
    "Type": "TransientModel",
    "Resources": {
        RESOURCES
    },
    "TimingParams": {
        TIMING_PARAMS
    },
    "SourceTimingControl": {
        SOURCE_TIMING_PARAMS
    }
}
```

Note

The value of the *TaskAlias* parameter can only contain English alphanumeric characters, hyphens, and underscores.

Note

The Control Software uses the configuration name, with the **_transient** string, for the *TaskAlias* parameter in thermal transient measurement tasks. For example, **"MyTestConfig_transient"**.

Note

SourceTimingControl is an optional parameter. If omitted, a default power on/off will be performed, with appropriate instrument delays.

Command example:


```
{
    "Command": "START_TASK",
    "TaskMode": "TRANSIENT",
    "ConfigName": "MyTestConfig",
   "TaskAlias": "MyTestConfig_transient",
   "Type": "TransientModel",
    "Resources": {
        "CurrentSourceParams": [{
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "S3Ch1",
                "OutputMode": "ON",
                "SetCurrent": 0.01,
                "VoltageCorner": 10
            }, {
                "Alias": "/T3STER/0/LP220/SLOT1/CH0",
                "UserAlias": "S1Ch1",
                "OutputMode": "PC",
                "SetCurrent": 0.1,
                "VoltageCorner": 2
            }
        ],
        "MeasCardChParams": [{
                "Sensitivity": [
                    0.002
                ],
                "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                "UserAlias": "S3Ch1",
               "PowerStep": "@POWERSTEP_DIODE;/T3STER/0/MS401/SLOT3/CH0;/
T3STER/0/LP220/SLOT1/CH0",
                "AutoRange": true,
                "RangeIdx": 13,
                "Uref": 0,
                "UrefSwitching": false,
                "UrefHeating": 0,
                "DividerAlias": ""
            }
        ]
    },
    "TimingParams": {
        "TransientMode": "Cooling",
        "HeatingTime": 15,
        "CoolingTime": 15,
        "DelayTime": 0,
        "SamplePerOctave": 1000,
        "Repeat": 1
    }
}
```


Answer message example:


```
{
     "Answer": "OK",
     "Message": ""
}
```

Stop a Thermal Transient Measurement Task [STOP_TASK - TRANSIENT]

This section provides a description of the **[STOP_TASK - TRANSIENT]** command.

Command format:

```
{
    "Command": "STOP_TASK",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "STOP_TASK",
    "TaskAlias": "MyTestConfig_transient"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Remove a Thermal Transient Measurement Task [REMOVE_TASK - TRANSIENT]

This section provides a description of the **[REMOVE_TASK - TRANSIENT]** command.

A task must be stopped or finished before it can be removed.

Command format:

```
{
    "Command": "REMOVE_TASK",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "REMOVE_TASK",
    "TaskAlias": "MyTestConfig_transient"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Task Related Queries

This section provides a description of task related queries.

Query the Status of a Task [QUERY_TASK_STATUS]

This section provides a description of the **[QUERY_TASK_STATUS]** command.

Command format:

```
{
    "Command": "QUERY_TASK_STATUS",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "QUERY_TASK_STATUS",
    "TaskAlias": "MyTestConfig_transient"
}
```

Answer message format:

```
{
     "ActualStep": Actual_step_number,
     "Answer": TASKSTATE,
     "Message": Message,
     "Percentage": Progress_of_task_in_percentage,
     "RemainingTime": Remaining_time_until_finish,
     "Status": Status_description,
     "TaskMode": TASKMODE,
     "TotalSteps": Steps_number
}
```

Variables:

- **Answer** (also called **TASKSTATE**):
  - *RUN*: the task is in progress; runs normally.
  - *FINISHED*: the task is finished successfully.
  - *STOPPED*: the task has been stopped (for example, stopped by the user).
  - *PENDING*: the task is pending (temporary state).
  - *INSUFFICIENT_RESOURCES*: the task cannot run because it has no access to its resources.
  - *ERROR*: something unexpected has happened.
  - *STOPPED_TIMEOUT*: the task has been stopped because a timeout value was reached (for example, thermostat stability timeout).
  - *STOPPED_INFO*: the task has been stopped due to an error that occurred (the reason why the task was stopped is described in the *Status* field).
  - *STOPPING*: the stopping sequence has been started for the task; the task will be stopped soon.
  - *WAIT_SAFETY_BTN*: the task is waiting for the user to push the HV enable button.
  - *WAIT_SAFETY_BTN_AND_VOLT_SEEK*: the task is waiting for the user to push the HV enable button.
- **TotalStep**: specifies the number of steps the measurement consists of.
- **ActualStep**: number of the step that is currently executed.
- **Percentage**: estimated progress of the task in percentage.
- **RemainingTime**: estimated time (in seconds) until the task is finished.
- **Status**: human-readable status message of the task.
- **Message**: message field, typically empty. In case an error has occurred, this field describes the problem).
- **TaskMode**:
  - *MONITORING_RESOURCE_ALLOCATION*: resources are allocated for further measurements.
  - *MONITORING*: monitoring is in progress.
  - *TSPCALIB*: TSP calibration is in progress.
  - *IDENTIFICATION*: identification is in progress.

- *TRANSIENT*: thermal transient measurement is in progress.

Answer message example:

```
{
     "ActualStep": 2,
     "Answer": "RUN",
     "Message": "",
     "Percentage": 13,
     "RemainingTime": 26,
     "Status": "Heating...",
     "TaskMode": "TRANSIENT",
     "TotalSteps": 3
}
```

Query the Partial Data of a Task [QUERY_TASK_PARTIAL_DATA]

This section provides a description of the **[QUERY_TASK_PARTIAL_DATA]** command.

This command can only be used in *TRANSIENT* and *TSPCALIB* mode.

Command format:

```
{
    "Command": "QUERY_TASK_PARTIAL_DATA",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "QUERY_TASK_PARTIAL_DATA",
    "TaskAlias": "MyTestConfig_transient"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Message": "",
     "Result": [
          {
               "ChAlias": Alias_of_the_measurement_channel,
               "x": [
                   x_values
               ],
               "y": [
                   y_values
               ]
          }
     ]
}
```

Variables:

- **x_values**):
  - For *TRANSIENT*: time data in seconds.
  - For *TSPCALIB*: temperature data in °C.
- **y_values**:
  - For *TRANSIENT*: calculated temperature data in °C.
  - For *TSPCALIB*: measured voltage data in V.

Query the Last Measured Data of a Task [QUERY_TASK_LAST_DATA]

This section provides a description of the **[QUERY_TASK_LAST_DATA]** command.

This command can only be used in *TRANSIENT*, *TSPCALIB*, and *MONITORING* mode.

In case you want to issue several queries for new data one after the other, wait at least 500ms between two queries.

Command format:

```
{
    "Command": "QUERY_TASK_LAST_DATA",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "QUERY_TASK_LAST_DATA",
    "TaskAlias": "MyTestConfig_transient"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Message": "",
     "Results": [
          {
               "Alias": Alias_of_resource,
               "Flags": [
                    Flags
               ],
               "Noise": Noise_value_in_volts,
               "Voltage": Voltage_value_in_volts,
               "Temperature": Temperature_in_degrees
          }
     ]
}
```

Variables:

- **Alias**: alias of a measurement or a thermometer channel.
- **Flags**: issues reported by the Hardware. For more information, see Query the Last Measured Data of a Task [QUERY_TASK_LAST_DATA].
- **Noise**: calculated noise value for a measurement channel (optional field).
- **Voltage**: voltage value measured by a measurement channel (optional field).
- **Temperature**: temperature value measured by a thermometer channel (optional field).

Answer message example:

```
{
     "Answer": "OK",
     "Message": "",
     "Results": [
          {
               "Alias": "/T3STER/0/MS401/SLOT3/CH0",
               "Flags": [
                    "Open Circuit"
               ],
               "Noise": null,
               "Voltage": 0.00000174493058745001
          }
     ]
}
```

Query the Links to the Result Files [QUERY_TASK_RESULT_FILE_LIST]

This section provides a description of the **[QUERY_TASK_RESULT_FILE_LIST]** command.

Command format:

```
{
    "Command": "QUERY_TASK_RESULT_FILE_LIST",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "QUERY_TASK_RESULT_FILE_LIST",
    "TaskAlias": "MyTestConfig_transient"
}
```

Answer message format:

```
{
    "Answer":"OK",
    "GlobalParameters":[
        {
            "Alias": Name_for_the_file,
            "Filename": Path_to_file
        }
    ],
    "Message":"",
    "Result":[
        {
            "ChAlias": Alias_of_the_measurement_channel,
            "Filename": Path_to_file,
            "PhaseType":"Heating" / "Cooling" / "TspCalib"
        },
        ...
    ]
}
```

Variables:

- **Filename**: link to the results file.

  To download the report, attach the link in the **Filename** field to the base URL of your system. For example, if you access the Control Software through *http://192.168.0.123:8085*, then the report in the example above can be downloaded from the following URL:

  http://192.168.0.123:8085/measurement/heating_MyTestConfig_transient_T3STER_1_MS401_SLOT3_CH1.par

- **ChAlias**:
  - For the classic file format, this field describes the alias of the measurement channel.
  - For the single file format (**.parx*), this field describes the *ConfigName*.

- **PhaseType**:
  - *Heating*
  - *Cooling*
  - *TspCalib*

Answer message example:


```
{
    "Answer":"OK",
    "GlobalParameters":[
        {
            "Alias":"Measurement Parameters",
            "Filename":"/measurement/
measparams_MyTestConfig_transient.json"
        }
    ],
    "Message":"",
    "Result":[
        {
            "ChAlias":"/T3STER/0/MS401/SLOT3/CH0",
            "Filename":"/measurement/
heating_MyTestConfig_transient_T3STER_1_MS401_SLOT3_CH1.par",
            "PhaseType":"Heating"
        },
        {
            "ChAlias":"/T3STER/0/MS401/SLOT3/CH0",
            "Filename":"/measurement/
heating_MyTestConfig_transient_T3STER_1_MS401_SLOT3_CH1.raw",
            "PhaseType":"Heating"
        },
        {
            "ChAlias":"/T3STER/0/MS401/SLOT3/CH0",
            "Filename":"/measurement/
cooling_MyTestConfig_transient_T3STER_1_MS401_SLOT3_CH1.par",
            "PhaseType":"Cooling"
        },
        {
            "ChAlias":"/T3STER/0/MS401/SLOT3/CH0",
            "Filename":"/measurement/
cooling_MyTestConfig_transient_T3STER_1_MS401_SLOT3_CH1.raw",
            "PhaseType":"Cooling"
        },
        {
            "ChAlias":"MyTestConfig",
      "Filename":"/measurement/cooling_MyTestConfig_transient.parx",
"PhaseType":"Cooling"
        },
        {
            "ChAlias":"MyTestConfig",
      "Filename":"/measurement/heating_MyTestConfig_transient.parx",
"PhaseType":"Heating"
        }
    ]
}
```

Query the Used Transient Model of the Task [QUERY_TASK_TRANSIENTMODEL]

This section provides a description of the **[QUERY_TASK_TRANSIENTMODEL]** command.

Command format:

```
{
    "Command": "QUERY_TASK_TRANSIENTMODEL",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "QUERY_TASK_TRANSIENTMODEL",
    "TaskAlias": "MyTestConfig_identification"
}
```

For the answer message format, see Base of the TransientModel.

Answer message example:


```
{
     "ConfigName": "MyTestConfig",
     "Resources": {
          "CurrentSourceParams": [
               {
                    "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                    "OutputMode": "ON",
                    "SetCurrent": 0.01,
                    "TriggerSource": "",
                    "UserAlias": "S3Ch1",
                    "VoltageCorner": 10
               },
               {
                    "Alias": "/T3STER/0/LP220/SLOT1/CH0",
                    "OutputMode": "PC",
                    "SetCurrent": 0.1,
                    "TriggerSource": "",
                    "UserAlias": "S1Ch1",
                    "VoltageCorner": 2
               }
          ],
          "MeasCardChParams": [
               {
                    "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                    "AutoRange": true,
                    "DividerAlias": "",
                    "PowerStep": "@POWERSTEP_DIODE;/T3STER/0/MS401/SLOT3/
CH0;/T3STER/0/LP220/SLOT1/CH0",
                    "RangeIdx": 13,
                    "Sensitivity": [
                         0.002
                    ],
                    "Uref": 0.5,
                    "UrefHeating": 0.5,
                    "UrefSwitching": false,
                    "UserAlias": "S3Ch1"
               }
          ]
     },
     "TimingParams": {
          "CoolingTime": 15,
          "DelayTime": 0,
          "HeatingTime": 15,
          "Repeat": 1,
          "SamplePerOctave": 1000,
          "TransientMode": "Cooling"
     },
     "Type": "TransientModel"
}
```

Query the Used TSP Calibration Model of the Task [QUERY_TASK_TSPCALIBRATIONMODEL]

This section provides a description of the **[QUERY_TASK_TSPCALIBRATIONMODEL]** command.

Command format:

```
{
    "Command": "QUERY_TASK_TSPCALIBRATIONMODEL",
    "TaskAlias": Name_of_the_task
}
```

Command example:

```
{
    "Command": "QUERY_TASK_TSPCALIBRATIONMODEL",
    "TaskAlias": "MyTestConfig_calibration"
}
```

For the answer message format, see Base of the TspCalibrationModel.

Answer message example:


```
{
     "ConfigName": "MyTestConfig",
     "Resources": {
          "CurrentSourceParams": [
               {
                    "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                    "OutputMode": "ON",
                    "SetCurrent": 0.01,
                    "TriggerSource": "",
                    "UserAlias": "S3Ch1",
                    "VoltageCorner": 10
               },
               {
                    "Alias": "/T3STER/0/LP220/SLOT1/CH0",
                    "OutputMode": "PC",
                    "SetCurrent": 0.1,
                    "TriggerSource": "",
                    "UserAlias": "S1Ch1",
                    "VoltageCorner": 2
               }
          ],
          "MeasCardChParams": [
               {
                    "Alias": "/T3STER/0/MS401/SLOT3/CH0",
                    "AutoRange": false,
                    "DividerAlias": "",
                    "RangeIdx": 13,
                    "Uref": 0.5,
                    "UrefHeating": 0,
                    "UrefSwitching": false,
                    "UserAlias": "S3Ch1"
               }
          ],
          "ThermostatParams": [
               {
                    "Alias": "/THERMOSTAT/0",
                    "StabilityCriteria": {
                         "DtMinMax": 1,
                         "DtTarget": 2,
                         "TimeWindow": 15,
                         "Timeout": 600
                    },
                    "UserAlias": "Th0"
               }
          ]
     },
     "TspCalibParams": {
          "CustomTemperature": 25,
          "DutStability": false,
          "EndAction": "StartTemp",
          "Mode": "Upwards",
          "ThtIntSensor": true,
          "Tmax": 85,
          "Tmin": 25,
          "Tstep": 10
```

```
        },
        "Type": "TspCalibrationModel"
    }
```

Query the List of Existing Tasks [QUERY_TASKLIST]

This section provides a description of the **[QUERY_TASKLIST]** command.

Command format:

```
{
    "Command": "QUERY_TASKLIST"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Message": "",
     "TaskList": [
          {
               "Percentage": Percentage_of_task_state,
               "TaskAlias": Name_of_the_task,
               "TaskMode": TASKMODE,
               "TaskState": TASKSTATE
          }
     ]
}
```

For more information on the variables, see Query the Status of a Task [QUERY_TASK_STATUS].

Answer message example:

```
{
     "Answer": "OK",
     "Message": "",
     "TaskList": [
          {
               "Percentage": 0,
               "TaskAlias": "MyTestConfig",
               "TaskMode": "MONITORING_RESOURCE_ALLOCATION",
               "TaskState": "RUN"
          }
     ]
}
```

Query the List of Existing Tasks Associated with a Specific Configuration [QUERY_CONFIGS_TASKLIST]

This section provides a description of the **[QUERY_CONFIGS_TASKLIST]** command.

Command format:

```
{
    "Command": "QUERY_CONFIGS_TASKLIST",
    "ConfigName": Name_of_the_config
}
```

Command example:

```
{
    "Command": "QUERY_CONFIGS_TASKLIST",
    "ConfigName": "MyTestConfig"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": "",
     "TaskAlias": [
          "MyTestConfig",
          "MyTestConfig_transient"
     ]
}
```

The *TaskAlias* will contain an array of the *TaskAliases* of all running tasks on the queried configuration.

Chapter 5
External Devices

This section provides a description of the commands related to external devices.

Reinitialize External Hardware [REINITIALIZE_EXTERNAL_HARDWARE]

In case you changed a hardware configuration, or added a new external device to the system, you must reinitialize the system with the new setup using the **[REINITIALIZE_EXTERNAL_HARDWARE]** command.

This section provides a description of the **[REINITIALIZE_EXTERNAL_HARDWARE]** command.

Command format:

```
{
    "Command": "REINITIALIZE_EXTERNAL_HARDWARE"
}
```

Answer message:

```
{
    "Answer": "OK",
    "Message": ""
}
```

Note

All commands related to external device manipulation/query are unavailable during initialization, which may take several seconds.

Note

This command will start a task with the **"HW DISCOVERY"** alias. You can check the progress of this task, using the QUERY_TASK_STATUS query.

This is a self-destruct task, so when the initialization is completed, the task will destroy itself. That means that if an initialization task cannot not be found, the task is finished successfully.

Thermostats

Thermostats must be connected to the T3STER SI RS232 port. Other ports are currently not supported.

Note

Currently, only one thermostat can be handled by the system on the RS232 port. The alias of the thermostat must be *"/THERMOSTAT/0"*.

Thermostat configurations contain the following variables:

- **BaudRate**: *1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 500000, 576000, 921600, 1000000, 1152000, 1500000, 2000000, 2500000, 3000000, 3500000*, or *4000000*.
- **DataBits**: *7* or *8*
- **Handshake**:
  - *NONE*: no handshake.
  - *XONXOFF*: Software flow control, also: XON/XOFF flow control.
  - *REQUESTTOSEND*: Hardware handshake, also: RTS/CTS flow control.
  - *REQUESTTOSENDXONXOFF*: both Software and Hardware handshakes.
- **Parity**:
  - *NONE*
  - *ODD*
  - *EVEN*
- **StopBits**:
  - *SB1*: 1 stop bit
  - *SB2*: 2 stop bits
- **ThermostatType**:
  - *JULABO_HE*: JULABO HE series protocol.
  - *JULABO_F*: JULABO F series protocol.
  - *JULABO_CF*: JULABO CF series protocol.
  - *MICREDTHT*: MICRED Dry Thermostat.
  - *ARROYO*: ARROYO Thermostat.
  - *ESPEC*: Espec Thermostat.

- *HUBER_PB*: Huber Thermostat with PB commands.

Get Thermostat Configuration [GET_THERMOSTAT_CONFIG]

This section provides a description of the **[GET_THERMOSTAT_CONFIG]** command.

To read the existing thermostat configuration, use the following command:

```
{
    "Command": "GET_THERMOSTAT_CONFIG",
    "Alias": "/THERMOSTAT/0"
}
```

Answer message format:

```
{
    "Answer": "OK",
    "Alias": "/THERMOSTAT/0",
    "SerialTransport": {
        "BaudRate": BAUDRATE,
        "DataBits": DATABITS,
        "Handshake": HANDSHAKE,
        "InterfaceID": "RS232",
        "Parity": PARITY,
        "StopBits": STOPBITS,
        "Timeout": Timeout_in_msec,
        "WriteSleep": Sleeping_time_after_every_command_in_msec
    },
    "StabilityCriteria": {
        "DtMinMax":
Default_value_for_maximum_allowed_temperature_change_in_degrees,
        "DtTarget": Default_value_for_DeltaT_from_target_in_degrees,
        "TimeWindow": Default_value_for_time_window_in_sec,
        "Timeout": Default_value_for_timeout_in_sec,
    },
    "ThermostatType": THERMOSTATTYPE
}
```

Answer message example:


```
{
     "Alias": "/THERMOSTAT/0",
     "Answer": "OK",
     "SerialTransport": {
          "BaudRate": 4800,
          "DataBits": 8,
          "Handshake": "NONE",
          "InterfaceID": "RS232",
          "Parity": "NONE",
          "StopBits": "ONE",
          "Timeout": 2000,
          "WriteSleep": 100
     },
     "StabilityCriteria": {
          "DtMinMax": 0.1,
          "DtTarget": 0.25,
          "TimeWindow": 60,
          "Timeout": 1800
     },
     "ThermostatType": "MICREDTHT"
}
```


Answer message example if the thermostat configuration does not exist:


```
{
     "Answer": "NOT_FOUND"
}
```


Save Thermostat Configuration [SAVE_THERMOSTAT_CONFIG]

This section provides a description of the **[SAVE_THERMOSTAT_CONFIG]** command.

To save a thermostat configuration, use the following command:

```
{
    "Command": "SAVE_THERMOSTAT_CONFIG",
    "Alias": "/THERMOSTAT/0",
    "SerialTransport": {
        "BaudRate": BAUDRATE,
        "DataBits": DATABITS,
        "Handshake": HANDSHAKE,
        "InterfaceID": "RS232",
        "Parity": PARITY,
        "StopBits": STOPBITS,
        "Timeout": Timeout_in_msec,
        "WriteSleep": Sleeping_time_after_every_command_in_msec
    },
    "StabilityCriteria": {
        "DtMinMax":
Default_value_for_maximum_allowed_temperature_change_in_degrees,
        "DtTarget": Default_value_for_DeltaT_from_target_in_degrees,
        "TimeWindow": Default_value_for_time_window_in_sec,
        "Timeout": Default_value_for_timeout_in_sec,
    },
    "ThermostatType": THERMOSTATTYPE
}
```

Note

Currently, only one thermostat can be handled by the system on the RS232 port. The alias of the thermostat must be *"/THERMOSTAT/0"*.

Command example:

```
{
     "Command": "SAVE_THERMOSTAT_CONFIG",
     "Alias": "/THERMOSTAT/0",
     "Answer": "OK",
     "SerialTransport": {
          "BaudRate": 4800,
          "DataBits": 8,
          "Handshake": "NONE",
          "InterfaceID": "RS232",
          "Parity": "NONE",
          "StopBits": "ONE",
          "Timeout": 2000,
          "WriteSleep": 100
     },
     "StabilityCriteria": {
          "DtMinMax": 0.1,
          "DtTarget": 0.25,
          "TimeWindow": 60,
          "Timeout": 1800
     },
     "ThermostatType": "MICREDTHT"
}
```

Answer message example:

```
{
     "Answer": "OK"
}
```

Note

After saving a new thermostat configuration, the system must be rebooted, or the Reinitialize External Hardware [REINITIALIZE_EXTERNAL_HARDWARE] command must be called to initialize the thermostat with the new parameters.

Enable Thermostat Circulator [ENABLE_THERMOSTAT]

This section provides a description of the **[ENABLE_THERMOSTAT]** command.

To enable a thermostat circulator, use the following command:

```
{
    "Command": "ENABLE_THERMOSTAT",
    "Alias": "/THERMOSTAT/0"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Disable Thermostat Circulator [DISABLE_THERMOSTAT]

This section provides a description of the **[DISABLE_THERMOSTAT]** command.

To disable a thermostat circulator, use the following command:

```
{
    "Command": "DISABLE_THERMOSTAT",
    "Alias": "/THERMOSTAT/0"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Set Thermostat Temperature [SET_THERMOSTAT_TEMPERATURE]

This section provides a description of the **[SET_THERMOSTAT_TEMPERATURE]** command.

To set the temperature of a thermostat manually, use the following command:

```
{
    "Command": "SET_THERMOSTAT_TEMPERATURE",
    "Alias": "/THERMOSTAT/0",
    "SetTemperature": Temperature_in_degrees
}
```

Command example:

```
{
    "Command": "SET_THERMOSTAT_TEMPERATURE",
    "Alias": "/THERMOSTAT/0",
    "SetTemperature": 12.34
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Query Thermostat Status [QUERY_THERMOSTAT_STATUS]

This section provides a description of the **[QUERY_THERMOSTAT_STATUS]** command.

To query the status of a thermostat manually, use the following command:

```
{
    "Command": "QUERY_THERMOSTAT_STATUS",
    "Alias": "/THERMOSTAT/0"
}
```

Answer message format:

```
{
    "ActualTemperature": Actual_temperature_in_degrees,
    "Answer": "OK",
    "Enabled": true/false,
    "HwStatus": Status_string_of_the_thermostat,
    "Message": "",
    "Status": STATUS,
    "TargetTemperature": Target_temperature_in_degrees
}
```

Answer message example:

```
{
     "ActualTemperature": 25.046,
     "Answer": "OK",
     "Enabled": true,
     "HwStatus": "03 REMOTE START",
     "Message": "",
     "Status": "STABLE",
     "TargetTemperature": 25
}
```

Possible values for the **Status** variable:

- *STABLE*: the thermostat is stabilized around the target temperature.
- *HEATING*: the thermostat is heating up.
- *COOLING*: the thermostat is cooling down.
- *APPROACHING*: the thermostat is approaching the target temperature by 1°C.
- *NOT STABLE*: the thermostat cannot reach stability before timeout.
- *UNKNOWN*: error status.

Query Unused Thermostat Ports [GET_ALL_UNUSED_PORTS_FOR_THERMOSTATS]

This section provides a description of the **[GET_ALL_UNUSED_PORTS_FOR_THERMOSTATS]** command.

To get a list of all ports which are not used for thermostats yet, use the following command:

```
{
    "Command": "GET_ALL_UNUSED_PORTS_FOR_THERMOSTATS"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Ports": [
          "RS232"
          "USB0"
     ]
}
```

Initialize New Thermostat [TRY_INIT_THEMOSTAT_AND_SAVE_CONFIG]

This section provides a description of the **[TRY_INIT_THERMOSTAT_AND_SAVE_CONFIG]** command.

To initialize a new thermostat and save the new - or modified - thermostat configuration, use the following command:

```
{
    "Command": "TRY_INIT_THEMOSTAT_AND_SAVE_CONFIG"
    "Alias": "/THERMOSTAT/0",
    "SerialTransport": {
        "BaudRate": BAUDRATE,
        "DataBits": DATABITS,
        "Handshake": HANDSHAKE,
        "InterfaceID": ID_of_the_port,
        "Parity": PARITY,
        "StopBits": STOPBITS,
        "Timeout": Timeout_in_msec,
        "WriteSleep": Sleeping_time_after_every_command_in_msec
    },
    "StabilityCriteria": {
        "DtMinMax":
Default_value_for_maximum_allowed_temperature_change_in_degrees,
        "DtTarget": Default_value_for_DeltaT_from_target_in_degrees,
        "TimeWindow": Default_value_for_time_window_in_sec,
        "Timeout": Default_value_for_timeout_in_sec,
    },
    "ThermostatType": THERMOSTATTYPE
}
```

When this command is issued, the system tries to initialize the thermostat with the specified parameters. If the process is successfully finished, the previous thermostat is removed and released from the system and the new thermostat configuration file is saved.

The *InterfaceID* can be any of the ports listed in the ports array returned by the **[GET_ALL_UNUSED_PORTS_FOR_THERMOSTATS]** command (see Query Unused Thermostat Ports [GET_ALL_UNUSED_PORTS_FOR_THERMOSTATS]).

Serial Communication Ports

The system has an RS485 and a USB-A port for external devices other than thermostats. Currently, only Simcenter Micred Boosters and TDK Genesys power supplies are supported.

If you want to attach more than one USB devices to the system, make sure that the hub is supplied by its own power supply.

Get a List of All Available Ports for External Devices [GET_ALL_AVAILABLE_PORTS]

This section provides a description of the **[GET_ALL_AVAILABLE_PORTS]** command.

To get a list of all available ports in the system, use the following command:

```
{
    "Command": "GET_ALL_AVAILABLE_PORTS"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Ports": Array_of_available_ports
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Ports": [
          "RS485",
          "USB0"
     ]
}
```

Query Unused Ports for External Devices [GET_ALL_UNUSED_PORTS_FOR_EXT_PSUS]

This section provides a description of the **[GET_ALL_UNUSED_PORTS_FOR_EXT_PSUS]** command.

To get a list of all ports that are not used for external devices yet, use the following command:

```
{
    "Command": "GET_ALL_UNUSED_PORTS_FOR_EXT_PSUS"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Ports": [
          "RS485"
     ]
}
```

Search for Devices on a Port [SEARCH_BUSINSTRUMENTS]

This section provides a description of the **[SEARCH_BUSINSTRUMENTS]** command.

To find the attached serial devices (for example, a TDK Genesys power supply) on a port, use the following command with the serial port parameters:

```
{
    "Command": "SEARCH_BUSINSTRUMENTS",
    "Ports": [
        {
            "BaudRate": BAUDRATE,
            "DataBits": DATABITS,
            "Handshake": HANDSHAKE,
            "InterfaceID": ID_of_the_port,
            "Parity": PARITY,
            "StopBits": STOPBITS,
            "Timeout": Timeout_in_msec,
            "WriteSleep": Sleeping_time_after_every_command_in_msec,
            "FromAddress": Lowest_address_number_for_searching,
            "ToAddress": Highest_address_number_for_searching
        }
    ]
}
```

The *InterfaceID* can be any port listed in the ports array of the Get a List of All Available Ports for External Devices [GET_ALL_AVAILABLE_PORTS] command.

The special variables in this command are the same as in thermostat configurations. For more information, see Thermostats.

Command example:

```
{
    "Command": "SEARCH_BUSINSTRUMENTS",
    "Ports": [
        {
            "BaudRate": 19200,
            "DataBits": 8,
            "Handshake": "NONE",
            "InterfaceID": "RS485",
            "Parity": "NONE",
            "StopBits": "ONE",
            "Timeout": 2000,
            "WriteSleep": -1,
            "FromAddress": 1,
            "ToAddress": 31
        }
    ]
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Ports": [
          {
               "Instruments": [
                    {
                         "Address": Address,
                         "InstrumentType": INSTRUMENT_TYPE,
                         "SerialNumber": Serial_number_of_the_device,
                         "Type": Exact_type_of_the_device
                    }
               ],
               "InterfaceID": "RS485"
          }
     ]
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Ports": [
          {
               "Instruments": [
                    {
                         "Address": 5,
                         "InstrumentType": "TDKPSU",
                         "SerialNumber": "123A456-7890",
                         "Type": "LAMBDA,GEN30-50"
                    },
                    {
                         "Address": 6,
                         "InstrumentType": "TDKPSU",
                         "SerialNumber": "123A456-7891",
                         "Type": "LAMBDA,GEN30-50"
                    }
               ],
               "InterfaceID": "RS485"
          }
     ]
}
```

Possible values of the **InstrumentType** parameter:

- *TDKPSU*: TDK power supply
- *HXMCG*: HXM CG - Current generator
- *HXMOS*: HXM OS - Output stage
- *HXMDIV*: HXM Divider

Add or Overwrite a Port Configuration [ADD_OR_OVERWRITE_BUS_TO_BUSINSTRUMENT_CONFIG]

This section provides a description of the **[ADD_OR_OVERWRITE_BUS_TO_BUSINSTRUMENT_CONFIG]** command.

To initialize an external device which was not handled automatically, you must save the port configuration first. The port configuration must contain the serial bus parameters, and the information about the devices on that bus.

To save a port configuration, use the following command.

```
{
     "Command": "ADD_OR_OVERWRITE_BUS_TO_BUSINSTRUMENT_CONFIG",
     "Buses": [
          {
               "Instruments": [
                    {
                         "Address": Address,
                         "Alias": Alias,
                         "InstrumentType": INSTRUMENT_TYPE,
                         "SerialNumber": Serial_number_of_the_device,
                         "Type": Exact_type_of_the_device
                    }
               ],
               "SerialTransport": {
                    "BaudRate": BAUDRATE,
                    "DataBits": DATABITS,
                    "Handshake": HANDSHAKE,
                    "InterfaceID": Port_name,
                    "Parity": PARITY,
                    "StopBits": STOPBITS,
                    "Timeout": Timeout_in_msec,
             "WriteSleep": Sleeping_time_after_every_command_in_msec
}
          }
     ]
}
```

The instrument parameters can be obtained using the Search for Devices on a Port [SEARCH_BUSINSTRUMENTS] command. The instrument alias can be any value, but it is suggested to use the serial number of the device.

Note

After saving a new port configuration, the system must be rebooted, or the Reinitialize External Hardware [REINITIALIZE_EXTERNAL_HARDWARE] command must be called to initialize the new devices.

Command example:

```
{
     "Command": "ADD_OR_OVERWRITE_BUS_TO_BUSINSTRUMENT_CONFIG",
     "Buses": [
          {
               "Instruments": [
                    {
                         "Address": 5,
                         "Alias": "123A456-7890",
                         "InstrumentType": "TDKPSU",
                         "SerialNumber": "123A456-7890",
                         "Type": "LAMBDA,GEN30-50"
                    },
                    {
                         "Address": 6,
                         "Alias": "123A456-7891",
                         "InstrumentType": "TDKPSU",
                         "SerialNumber": "123A456-7891",
                         "Type": "LAMBDA,GEN30-50"
                    }
               ],
               "SerialTransport": {
                    "BaudRate": 19200,
                    "DataBits": 8,
                    "Handshake": "NONE",
                    "InterfaceID": "RS485",
                    "Parity": "NONE",
                    "StopBits": "NONE",
                    "Timeout": 2000,
                    "WriteSleep": -1
               }
          }
     ]
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Query an Existing Port Configuration [QUERY_BUS_FROM_BUSINSTRUMENT_CONFIG]

This section provides a description of the **[QUERY_BUS_FROM_BUSINSTRUMENT_CONFIG]** command.

To query an existing port configuration by the name of the port, use the following command:

```
{
    "Command": "QUERY_BUS_FROM_BUSINSTRUMENT_CONFIG",
    "BusId": Port_name
}
```

Command example:

```
{
    "Command": "QUERY_BUS_FROM_BUSINSTRUMENT_CONFIG",
    "BusId": "RS485"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Buses": [
          {
               "Instruments": [
                    {
                         "Address": Address,
                         "Alias": Alias,
                         "InstrumentType": INSTRUMENT_TYPE,
                         "SerialNumber": Serial_number_of_the_device,
                         "Type": Exact_type_of_the_device
                    }
               ],
               "SerialTransport": {
                    "BaudRate": BAUDRATE,
                    "DataBits": DATABITS,
                    "Handshake": HANDSHAKE,
                    "InterfaceID": Port_name,
                    "Parity": PARITY,
                    "StopBits": STOPBITS,
                    "Timeout": Timeout_in_msec,
             "WriteSleep": Sleeping_time_after_every_command_in_msec
},
          }
     ],
     "Message": ""
}
```

Answer message example:


```
{
     "Answer": "OK",
     "Buses": [
          {
               "Instruments": [
                    {
                         "Address": 5,
                         "Alias": "123A456-7890",
                         "InstrumentType": "TDKPSU",
                         "SerialNumber": "123A456-7890",
                         "Type": "LAMBDA,GEN30-50"
                    }
               ],
               "SerialTransport": {
                    "BaudRate": 19200,
                    "DataBits": 8,
                    "Handshake": "NONE",
                    "InterfaceID": "RS485",
                    "Parity": "NONE",
                    "StopBits": "NONE",
                    "Timeout": 2000,
                    "WriteSleep": -1
               }
          }
     ],
     "Message": ""
}
```


Get All Saved Port Configurations [GET_BUSINSTRUMENT_INFO]

This section provides a description of the **[GET_BUSINSTRUMENT_INFO]** command.

To query all existing port configurations and their availability, use the following command:


```
{
    "Command": "GET_BUSINSTRUMENT_INFO"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Buses": [
          {
               "Available": true/false,
               "BusId": Port_name,
               "Instruments": [
                    {
                         "Address": Address,
                         "Alias": Alias,
                         "Available": true/false,
                         "InstrumentType": INSTRUMENT_TYPE,
                         "SerialNumber": Serial_number_of_the_device,
                         "Type": Exact_type_of_the_device
                    }
               ],
               "SerialTransport": {
                    "BaudRate": BAUDRATE,
                    "DataBits": DATABITS,
                    "Handshake": HANDSHAKE,
                    "InterfaceID": Port_name,
                    "Parity": PARITY,
                    "StopBits": STOPBITS,
                    "Timeout": Timeout_in_msec,
             "WriteSleep": Sleeping_time_after_every_command_in_msec
},
               "Status": "OK"
          }
     ],
     "Message": ""
}
```

Answer message example:


```
{
     "Answer": "OK",
     "Buses": [
          {
               "Available": true,
               "BusId": "RS485",
               "Instruments": [
                    {
                         "Address": 5,
                         "Alias": "123A456-7890",
                         "Available": true,
                         "InstrumentType": "TDKPSU",
                         "SerialNumber": "123A456-7890",
                         "Type": "LAMBDA,GEN30-50"
                    }
               ],
               "SerialTransport": {
                    "BaudRate": 19200,
                    "DataBits": 8,
                    "Handshake": "NONE",
                    "InterfaceID": "RS485",
                    "Parity": "NONE",
                    "StopBits": "NONE",
                    "Timeout": 2000,
                    "WriteSleep": -1
               },
               "Status": "OK"
          }
     ],
     "Message": ""
}
```


Remove an Existing Port Configuration [REMOVE_BUS_FROM_BUSINSTRUMENT_CONFIG]

This section provides a description of the **[REMOVE_BUS_FROM_BUSINSTRUMENT_CONFIG]** command.

To remove an existing port configuration, use the following command:

```
{
    "Command": "REMOVE_BUS_FROM_BUSINSTRUMENT_CONFIG",
    "BusId": "RS485"
}
```

Answer message:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Get a List of All Attachable PSUs [QUERY_EXTERNAL_ATTACHABLE_PSUS]

This section provides a description of the **[QUERY_EXTERNAL_ATTACHABLE_PSUS]** command.

To query the list of all available external power supply units, use the following command:

```
{
    "Command": "QUERY_EXTERNAL_ATTACHABLE_PSUS"
}
```

Answer message format:

```
{
    "Answer": "OK",
    "ExternalPsu": [
        {
            "Address": Address,
            "Alias": Alias,
            "Port": Port_name,
            "SerialNumber": Serial_number_of_the_PSU,
            "Type": Type_of_the_PSU
        }
    ],
    "Message": ""
}
```

Answer message example:


```
{
    "Answer": "OK",
    "ExternalPsu": [
        {
            "Address": 5,
            "Alias": "/EXTSUPPLY/123A456-7890",
            "Port": "RS485",
            "SerialNumber": "123A456-7890",
            "Type": "LAMBDA,GEN30-50"
        },
        {
            "Address": 6,
            "Alias": "/EXTSUPPLY/123A456-7891",
            "Port": "RS485",
            "SerialNumber": "123A456-7891",
            "Type": "LAMBDA,GEN30-50"
        }
    ],
    "Message": ""
}
```

Boosters

This section provides a description of the commands related to the management of Boosters and their associated PSUs.

Note

PWB 240A type boosters are assigned to their power supplies automatically by the system, thus, these boosters cannot be configured manually.

Query all Initiated USB Booster Devices [QUERY_EXTERNAL_DEVICES]

This section provides a description the **[QUERY_EXTERNAL_DEVICES]** command.

To query all initiated USB boosters in the system, use the following command:

```
{
    "Command": "QUERY_EXTERNAL_DEVICES"
}
```

Answer message format:

```
{
    "Answer": "OK",
    "Devices": [
        {
            "Manufacturer": "Mentor Graphics",
            "Port": Port_name,
            "ProductId": Product_id,
            "ProductName": Product_name,
            "SerialNumber": Serial_number_of_the_device,
            "Type": Type_of_the_device,
            "VendorId": Vendor_id
        }
    ],
    "Message": ""
}
```

Answer message format:

```
{
    "Answer": "OK",
    "Devices": [
        {
            "Manufacturer": "Mentor Graphics",
            "Port": "USB0",
            "ProductId": "cb3a",
            "ProductName": "Booster",
            "SerialNumber": "BLD12345",
            "Type": "BLD50V30",
            "VendorId": "0403"
        }
    ],
    "Message": ""
}
```

Save a Booster Configuration [SAVE_BOOSTER_CONFIG]

This section provides a description the **[SAVE_BOOSTER_CONFIG]** command.

To save a booster configuration, use the following command:

```
{
    "Command": "SAVE_BOOSTER_CONFIG",
    "ExtPsuAliases": [
        Alias_of_the_PSU_attached_to_first_channel,
        Alias_of_the_PSU_attached_to_second_channel
    ],
    "SerialNumber": Serial_number_of_the_booster
}
```

Note

If a configuration already exits with the same booster (same serial number), it will be overwritten.

Command example:

```
{
    "Command": "SAVE_BOOSTER_CONFIG",
    "ExtPsuAliases": [
        "/EXTSUPPLY/123A456-7890",
        "/EXTSUPPLY/123A456-7891"
    ],
    "SerialNumber": "BLD123456"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Get Booster Configuration List [GET_BOOSTER_CONFIG_LIST]

This section provides a description the **[GET_BOOSTER_CONFIG_LIST]** command.

To query all available booster configurations, use the following command:

```
{
    "Command": "GET_BOOSTER_CONFIG_LIST"
}
```

Answer message format:

```
{
    "Answer": "OK",
    "BoosterPsuAssociations": [
        {
            "ExtPsuAliases": [
                Alias_of_the_PSU_attached_to_first_channel,
                Alias_of_the_PSU_attached_to_second_channel
            ],
            "SerialNumber": Serial_number_of_the_booster
        }
    ],
    "Message": ""
}
```

Answer message example:

```
{
     "Answer": "OK",
     "BoosterPsuAssociations": [
          {
               "ExtPsuAliases": [
                    "/EXTSUPPLY/123A456-7890",
                    "/EXTSUPPLY/123A456-7891"
               ],
               "SerialNumber": "BLD123456"
          },
          {
               "ExtPsuAliases": [
                    "/EXTSUPPLY/123A456-7892",
                    "/EXTSUPPLY/123A456-7893"
               ],
               "SerialNumber": "BLD123457"
          }
     ],
     "Message": ""
}
```

Get Booster Configuration List with PSU Availability Information [GET_BOOSTER_CONFIG_LIST_EXTENDED]

This section provides a description the **[GET_BOOSTER_CONFIG_LIST_EXTENDED]** command.

To query all available booster configurations with PSU availability, use the following command:

```
{
    "Command": "GET_BOOSTER_CONFIG_LIST_EXTENDED"
}
```

Answer message format:

```
{
    "Answer": "OK",
    "BoosterPsuAssociations": [
        {
            "Available": [
                Availability_of_the_PSU_attached_to_first_channel true/
false,
                Availability_of_the_PSU_attached_to_first_channel true/
false
            ],
            "ExtPsuAliases": [
                Alias_of_the_PSU_attached_to_first_channel,
                Alias_of_the_PSU_attached_to_second_channel
            ],
            "SerialNumber": Serial_number_of_the_booster
        }
    ],
    "Message": ""
}
```

Answer message example:

```
{
    "Answer": "OK",
    "BoosterPsuAssociations": [
        {
            "Available": [
                true,
                true
            ],
            "ExtPsuAliases": [
                "/EXTSUPPLY/836A032-1236",
                "/EXTSUPPLY/836A032-1237"
            ],
            "SerialNumber": "BLD10232"
        }
    ],
    "Message": ""
}
```

Remove a Booster Configuration [REMOVE_BOOSTER_CONFIG]

This section provides a description the **[REMOVE_BOOSTER_CONFIG]** command.

To remove an existing booster configuration, use the following command:

```
{
    "Command": "REMOVE_BOOSTER_CONFIG",
    "SerialNumber": Serial_number_of_the_booster
}
```

Command example:

```
{
    "Command": "REMOVE_BOOSTER_CONFIG",
    "SerialNumber": "BLD00627"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Modular Devices

This section provides a description of the commands related to the management of modular devices.

Save a Modular Device Configuration [SAVE_HXMBOOSTER_CONFIG]

This section provides a description of the **[SAVE_HXMBOOSTER_CONFIG]** command.

To save a modular device configuration, use the following command:

```
{
    "Command": "SAVE_HXMBOOSTER_CONFIG",
    "Alias": Alias_of_the_configuration,
    "ExtPsuAlias": Alias_of_attached_PSU,
    "SerialNumberCg": Serial_number_of_the_CG,
    "SerialNumberOs": Serial_number_of_the_OS,
    "SerialNumberDiv": Serial_number_of_the_DIVIDER
}
```

Note

If a configuration already exits with the same modular device (same serial number), it will be overwritten.

Add a new configuration to the end of the configuration list. If a record already exists with the same alias or the same devices, it will be overwritten.

Command example:

```
{
    "Command": "SAVE_HXMBOOSTER_CONFIG",
    "Alias": "MyModularConfig",
    "ExtPsuAlias": "/EXTSUPPLY/123A456-7890",
    "SerialNumberCg": "HXMCG123456",
    "SerialNumberOs": "HXMOS123456",
    "SerialNumberDiv": "HXMDIV123456"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Get a List of All Modular Device Configurations [GET_HXMBOOSTER_CONFIG_LIST]

This section provides a description of the **[GET_HXMBOOSTER_CONFIG_LIST]** command.

To query all available modular device configurations, use the following command:

```
{
    "Command": "GET_HXMBOOSTER_CONFIG_LIST"
}
```

Answer message format:

```
{
    "Answer": "OK",
    "HxmBoosterAssociations": [
        {
            "Alias": Alias_of_the_configuration,
            "ExtPsuAlias": Alias_of_attached_PSU,
            "SerialNumberCg": Serial_number_of_the_CG,
            "SerialNumberOs": Serial_number_of_the_OS,
            "SerialNumberDiv": Serial_number_of_the_DIVIDER
        }
    ],
    "Message": ""
}
```

Answer message example:

```
{
     "Answer": "OK",
     "HxmBoosterAssociations": [
          {
               "Alias": "MyModularConfig",
               "ExtPsuAlias": "/EXTSUPPLY/211B028-0002",
               "SerialNumberCg": "HXMCG123456",
               "SerialNumberOs": "HXMOS123456",
               "SerialNumberDiv": "HXMDIV123456"
          }
     ],
     "Message": ""
}
```

Get Modular Device Configuration List with PSU Availability Information [GET_HXMBOOSTER_CONFIG_LIST_EXTENDED]

This section provides a description the **[GET_HXMBOOSTER_CONFIG_LIST_EXTENDED]** command.

To query all available modular device configurations with PSU availability, use the following command:

```
{
    "Command": "GET_HXMBOOSTER_CONFIG_LIST_EXTENDED"
}
```

Answer message format:

```
{
    "Answer": "OK",
    "HxmBoosterAssociations": [
        {
            "Alias": Alias_of_the_configuration,
            "Available": true/false,
            "ExtPsuAlias": Alias_of_attached_PSU,
            "SerialNumberCg": Serial_number_of_the_CG,
         "SerialNumberOs": Serial_number_of_the_OS,
         "SerialNumberDiv": Serial_number_of_the_DIVIDER
        }
    ],
    "Message": ""
}
```

Answer message example:

```
{
    "Answer": "OK",
    "HxmBoosterAssociations": [
        {
            "Alias": "MyModularConfig",
            "Available": true,
            "ExtPsuAlias": "/EXTSUPPLY/123A456-7890",
            "SerialNumberCg": "HXMCG123456",
            "SerialNumberOs": "HXMOS123456",
            "SerialNumberDiv": "HXMDIV123456"
        }
    ],
    "Message": ""
}
```

Remove a Modular Device Configuration [REMOVE_HXMBOOSTER_CONFIG]

This section provides a description the **[REMOVE_HXMBOOSTER_CONFIG]** command.

To remove an existing modular device configuration, use the following command:

```
{
    "Command": "REMOVE_HXMBOOSTER_CONFIG",
    "Alias": Alias_of_the_configuration
}
```

Command example:

```
{
    "Command": "REMOVE_HXMBOOSTER_CONFIG",
    "Alias": "MyModularConfig"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Chapter 6
Global System Settings

This section provides a description of the commands related to global system settings.

Identification Settings

The Simcenter Micred T3STER SI has the ability to determine the appropriate measurement range and reference voltage for DUT measurements.

When the *AutoRange* property is set to **true** for a measurement channel (see Measurement Channels in a Configuration [MeasCardChParams]), an identification will be performed before the measurement with the identification parameters described in this section.

Get the Identification Parameters [GET_IDENTIFICATION_SETTINGS]

This section provides a description of the **[GET_IDENTIFICATION_SETTINGS]** command.

To query identification parameters, use the following command:

```
{
    "Command": "GET_IDENTIFICATION_SETTINGS"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Identification": {
          "ExpectedMaxTemperatureChange":
Maximum_temperature_change_in_degrees,
          "HeatingPulseLength":
Heating_pulse_length_of_the_identification_in_msec,
          "Margin": Margin_for_overhead,
          "Time1ApproxPoint": Approximation_start_point_in_msec,
          "Time2ApproxPoint": Approximation_end_point_in_msec
     },
     "Message": ""
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Identification": {
          "ExpectedMaxTemperatureChange": 200,
          "HeatingPulseLength": 20,
          "Margin": 2,
          "Time1ApproxPoint": 10,
          "Time2ApproxPoint": 20
     },
     "Message": ""
}
```

Set the Identification Parameters [SET_IDENTIFICATION_SETTINGS]

This section provides a description of the **[SET_IDENTIFICATION_SETTINGS]** command.

To set the identification parameters, use the following command:

```
{
    "Command": "SET_IDENTIFICATION_SETTINGS",
    "Identification": {
        "ExpectedMaxTemperatureChange":
Maximum_temperature_change_in_degrees,
        "HeatingPulseLength":
Heating_pulse_length_of_the_identification_in_msec,
        "Margin": Margin_for_overhead,
        "Time1ApproxPoint": Approximation_start_point_in_msec,
        "Time2ApproxPoint": Approximation_end_point_in_msec
    },
}
```

Command example:

```
{
    "Command": "SET_IDENTIFICATION_SETTINGS",
    "Identification": {
        "ExpectedMaxTemperatureChange": 200,
        "HeatingPulseLength": 20,
        "Margin": 2,
        "Time1ApproxPoint": 10,
        "Time2ApproxPoint": 15
    }
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Global GUI Settings

This section provides a description of the global GUI settings.

The following global GUI settings are available:

- **FileformatGeneration**: generates the measurement results in the specified file format for downloading. It only affects the output before the measurement is started.

  The following file formats are available:

  - *"Classic"*: classic format (**.par* and **.raw*)
  - *"Parx"*: single file format (**.parx*)
  - *"Both"*: generates both file formats

- **SwitchToPlot**: switches to *Plot* view automatically when the measurement is started (this is a GUI feature only; it does not affect the API).

Get Global GUI Settings [GET_GUI_SETTINGS]

This section provides a description of the **[GET_GUI_SETTINGS]** command.

To get the global GUI settings in the system, use the following command:

```
{
    "Command":"GET_GUI_SETTINGS"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "GuiGlobalSettings": {
          "FileformatGeneration": File_format,
          "SwitchToPlot": true/false
     },
     "Message": ""
}
```

Answer message example:

```
{
     "Answer": "OK",
     "GuiGlobalSettings": {
          "FileformatGeneration": "Both",
          "SwitchToPlot": true
     },
     "Message": ""
}
```

Set Global GUI Settings [SET_GUI_SETTINGS]

This section provides a description of the **[SET_GUI_SETTINGS]** command.

To set the global GUI parameters, use the following command:

```
{
    "Command": "SET_GUI_SETTINGS",
    "GuiGlobalSettings": {
        "FileformatGeneration": File_format,
        "SwitchToPlot": true/false
    }
}
```

For more information on the supported file formats, see Global GUI Settings.

Command example:

```
{
     "Command": "SET_GUI_SETTINGS",
     "GuiGlobalSettings": {
          "FileformatGeneration": "Both"
     }
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": ""
}
```

Chapter 7
Miscellaneous Commands

This section provides a description of miscellaneous commands.

Check LP220 Parameters [QUERY_LP220_PARAMETERS_VALIDITY]

This section provides a description of the [QUERY_LP220_PARAMETERS_VALIDITY] command.

Using this command, you can perform an LP220 parameter check with a human-readable error description. The reply contains a merged error list, where the errors are separated by *'\r\n'*.

Command format:

```
{
    "Command": "QUERY_LP220_PARAMETERS_VALIDITY",
    "CurrentCornerCh0Min":
LP220_first_channel_current_minimum_value_in_ampere,
    "CurrentCornerCh0Max":
LP220_first_channel_current_maximum_value_in_ampere,
    "CurrentCornerCh1Min":
LP220_second_channel_current_minimum_value_in_ampere,
    "CurrentCornerCh1Max":
LP220_second_channel_current_maximum_value_in_ampere,
    "VoltageCornerCh0Min":
LP220_first_channel_voltage_minimum_value_in_volt,
    "VoltageCornerCh0Max":
LP220_first_channel_voltage_maximum_value_in_volt,
    "VoltageCornerCh1Min":
LP220_second_channel_voltage_minimum_value_in_volt,
    "VoltageCornerCh1Max":
LP220_second_channel_voltage_maximum_value_in_volt
}
```

Command example:

```
{
    "Command": "QUERY_LP220_PARAMETERS_VALIDITY",
    "CurrentCornerCh0Min": 0.0,
    "CurrentCornerCh0Max": 1.0,
    "CurrentCornerCh1Min": 0.0,
    "CurrentCornerCh1Max": 1.0,
    "VoltageCornerCh0Min": 0.0,
    "VoltageCornerCh0Max": 35.0,
    "VoltageCornerCh1Min": 0.0,
    "VoltageCornerCh1Max": 35.0
}
```

Answer message example:

```
{
     "Answer": "OUT_OF_LIMITS",
     "Message": "LP220 CH1: Current 1A is out of limit (the allowed value
is 0.5A if voltage is 35V)\r\nLP220 CH1: Voltage 35V is out of limit (the
allowed value is 20V if current is 1A)\r\nLP220 CH2: Current 1A is out of
limit (the allowed value is 0.5A if voltage is 35V)\r\nLP220 CH2: Voltage
35V is out of limit (the allowed value is 20V if current is 1A)"
}
```

Query Thermometer Channels Standard Equations [QUERY_THERMOMETER_STANDARD_EQUATIONS]

This section provides a description of the **[QUERY_THERMOMETER_STANDARD_EQUATIONS]** command.

To query thermometer channels standard equations, use the following command:

```
{
    "Command":"QUERY_THERMOMETER_STANDARD_EQUATIONS"
}
```

Answer message format:

```
{
     "Answer": "OK",
     "Message": "",
     "ThermometerStandardEquations": [{
        "Name": SENSOR_TYPE,
        "Coefficients": [
            list of coefficients, highest index is the highest polynom
coefficient
        ],
        "Type": "Polynomial"
     }]
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": "",
     "ThermometerStandardEquations": [
          {
               "Coefficients": [
                    -247.29,
                    2399.2,
                    639.62,
                    1024.1
               ],
               "Name": "PT100",
               "Type": "Polynomial"
          }, ...
     ],
}
```

Which means:

$$T_{PT100} = -247.29 + 2399.2R + 639.62R^2 + 1024.1R^3$$

Appendix 8
Appendix

This section provides a description of power step calculations, enums formats, measurement and thermometer channels range definitions, and application card Hardware flags.

Power Step Calculations

This section provides a description of power step calculations.

Query Available Power Step Equations [GET_POWERSTEP_EQUATIONS_FORMAT]

To query the currently available power step equations and their formats, use the following command:

```
{
    "Command":"GET_POWERSTEP_EQUATIONS_FORMAT"
}
```

Answer message example:

```
{
     "Answer": "OK",
     "Message": "",
     "Powersteps": [
          "0",

"@POWERSTEP_DIODE;SensorCurrentSourceAlias;DriveCurrentSourceAlias",
          "@POWERSTEP_PASSIVE;MeasChannelAlias0;MeasChannelAlias1;...",

"@POWERSTEP_THREEPOLE_BASE_GATE_GROUNDED_VJUMP;DriveCurrentSourceAlias;Vo
ltageSourceAlias",

"@POWERSTEP_THREEPOLE_BASE_GATE_GROUNDED_IJUMP;SensorCurrentSourceAlias;D
riveCurrentSourceAlias;VoltageSourceAlias",

"@POWERSTEP_THREEPOLE_ON_STATE;SensorCurrentSourceAlias;DriveCurrentSourc
eAlias",

"@POWERSTEP_THREEPOLE_DIODE_ON_STATE_HEATING;SensorCurrentSourceAlias;Dri
veCurrentSourceAlias",

"@POWERSTEP_RDSON;SensorCurrentSourceAlias;DriveCurrentSourceAlias;RdsonG
eneratorAlias"
     ]
}
```

Diode Power Step

For simple diode measurements, use the following format:

```
"@POWERSTEP_DIODE;SensorCurrentSourceAlias;DriveCurrentSourceAlias"
```

For example:

```
"@POWERSTEP_DIODE;/T3STER/0/MS401/SLOT3/CH0;/T3STER/0/LP220/SLOT1/CH0"
```

Variables:

- *SensorCurrentSourceAlias*: the alias of the I_{sense}
- *DriveCurrentSourceAlias*: the alias of the I_{drive}

Passive Power Step

For passive elements, the power step is calculated as the sum of the power steps of other measurement channels:

```
"@POWERSTEP_PASSIVE;MeasChannelAlias0;MeasChannelAlias1;..."
```

For example:

```
"@POWERSTEP_PASSIVE;/T3STER/0/MS401/SLOT3/CH0;/T3STER/0/MS401/SLOT3/CH1"
```

Variables:

- *MeasChannelAlias0*: the alias of the other measurement channels

3-Pole Base (Gate) Grounded Setup, Voltage Jump Method Power Step

For the 3-pole base (gate) grounded setup, voltage jump method, use the following format:

```
"@POWERSTEP_THREEPOLE_BASE_GATE_GROUNDED_VJUMP;DriveCurrentSourceAlias;Vo
ltageSourceAlias"
```

For example:

```
"@POWERSTEP_THREEPOLE_BASE_GATE_GROUNDED_VJUMP;/T3STER/0/LP220/SLOT1/
CH0;/T3STER/0/LP220/SLOT1/CH1"
```

Variables:

- *DriveCurrentSourceAlias*: the alias of the I_{drive}
- *VoltageSourceAlias*: the alias of the V_{CB}

3-Pole Base (Gate) Grounded Setup, Current Jump Method Power Step

For the 3-pole base (gate) grounded setup, current jump method, use the following format:

```
"POWERSTEP_THREEPOLE_BASE_GATE_GROUNDED_IJUMP;SensorCurrentSourceAlias;Dr
iveCurrentSourceAlias;VoltageSourceAlias"
```

For example:

```
"@POWERSTEP_THREEPOLE_BASE_GATE_GROUNDED_IJUMP;/T3STER/0/MS401/SLOT3/
CH0;/T3STER/0/LP220/SLOT1/CH0;/T3STER/0/LP220/SLOT1/CH1"
```

Variables:

- *SensorCurrentSourceAlias*: the alias of the I_{sense}
- *DriveCurrentSourceAlias*: the alias of the I_{drive}
- *VoltageSourceAlias*: the alias of the V_{CB}

3-Pole On State Setup Power Step

For the 3-pole on state setup, use the following format:

```
"@POWERSTEP_THREEPOLE_ON_STATE;SensorCurrentSourceAlias;DriveCurrentSourc
eAlias"
```

For example:

```
"@POWERSTEP_THREEPOLE_ON_STATE;/T3STER/0/MS401/SLOT3/CH0;/T3STER/0/LP220/
SLOT1/CH0"
```

Variables:

- *SensorCurrentSourceAlias*: the alias of the I_{sense}
- *DriveCurrentSourceAlias*: the alias of the I_{drive}

3-Pole On State Heating, Body Diode Measurement Setup Power Step

For the 3-pole on state heating, body diode measurement setup, use the following format:

```
"@POWERSTEP_THREEPOLE_DIODE_ON_STATE_HEATING;SensorCurrentSourceAlias;Dri
veCurrentSourceAlias"
```

For example:

```
"@POWERSTEP_THREEPOLE_DIODE_ON_STATE_HEATING;/T3STER/0/MS401/SLOT3/CH0;/
T3STER/0/LP220/SLOT1/CH0"
```

Variables:

- *SensorCurrentSourceAlias*: the alias of the I_{sense}
- *DriveCurrentSourceAlias*: the alias of the I_{drive}

RDS-ON Power Step

For the RDS-ON measurement setup, use the following format:

```
"@POWERSTEP_RDSON;SensorCurrentSourceAlias;DriveCurrentSourceAlias;RdsonG
eneratorAlias"
```

For example:

```
"@POWERSTEP_RDSON;/T3STER/0/MS401/SLOT3/CH0;/T3STER/0/LP220/SLOT1/CH0;/
T3STER/0/MS401/SLOT3/CH1"
```

Variables:

- *SensorCurrentSourceAlias*: the alias of the I_{sense}

- *DriveCurrentSourceAlias*: the alias of the I_{drive}
- *RdsonGeneratorAlias*: as the V_{DS} is currently measured with a measurement channel, use the alias of the measurement channel, where the V_{DS} is measured.

Enums Format

This section provides a description of the enums format.

Enums Related to Current Sources

OUTPUT_MODE:

- *"ON"*: always on
- *"OFF"*: always off
- *"PC"*: switching (ON for heating and OFF for cooling)

Enums Related to Current Sources with Active Load

ACTIVELOAD_PROGRAMMING_MODE:

- *"FULL_PARAMS"*: parameters set manually (requires the *OnStateDutVoltage* field)
- *"AUTO_VOLTAGE_CORNER_SEEK"*: the **Auto voltage corner seek** option is selected (the on state voltage will be detected automatically; requires the *PulseLengthMs* field)

Enums Related to Trigger Outputs

TRIGGER_OUTPUT_MODE:

- *"High"*: always in High state
- *"Low"*: always in Low state
- *"Switched"*: High state for heating and Low state for cooling
- *"Switched Inverted"*: Low state for heating and High state for cooling
- *"Disabled"*: disabled (high-Z)

Enums Related to Voltage Sources

OUTPUT_MODE:

- *"ON"*: always on (requires the *OnStateVoltage* field)
- *"OFF"*: always off

- *"SWITCHVOLT"*: switching between two states (requires the *OnStateVoltage* and *OffStateVoltage* fields)

Enums Related to TSP Calibration

END_ACTION:

- *"StartTemp"*: when the calibration is finished, the thermostat is set to the start temperature of the calibration.
- *"EndTemp"*: when the calibration is finished, the thermostat is set to the last measured temperature during the calibration.
- *"CustomTemp"*: when the calibration is finished, the thermostat is set to the temperature described in the *CustomTemperature* field.

CALIBRATION_DIRECTION:

- *"Upwards"*: from T_{min} to T_{max}
- *"Downwards"*: from T_{max} to T_{min}

Enums Related to Source Timing Control

TYPE:

- *"CurrentSource"*: addresses a current source
- *"CurrentSourceWithActiveload"*: addresses a current source with active load
- *"VoltageSource"*: addresses a voltage source

Enums Related to Any Source with Delay

DELAY_MODE:

- *"NOT_SUPPORTED"*: for this source, the delay option is not supported
- *"CONTINUOUS"*: for this source, delay is only available for the falling trigger
- *"CONTINUOUS_RISING_FALLING"*: for this source, delay is available both for the rising and for the falling trigger
- *"DISCRETE"*: for this source, delay is only available for the falling trigger and it only accepts discrete values
- *"DISCRETE_RISING_FALLING"*: for this source, delay is available both for the rising and for the falling trigger and it only accepts discrete values

Measurement Channels Range Definitions

This section provides a description of the measurement channels range definitions.

Table 8-1. Measurement Channels Range Definitions

Idx	Gain Mux	Gain	Atten	Nominal FS Range [V]	Vin [V] min-max	Total Noise RTI [µV]
0	GainX5_Offset	5	0.125	32	±80	329
1	GainX10_Offset	5	0.125	16	±80	240
2	GainX20_Offset	5	0.125	8	±80	212
3	GainX2_Offset	5	0.25	40	±40	360
4	GainX5_Offset	5	0.25	16	±40	204
5	GainX10_Offset	5	0.25	8	±40	170
6	GainX2_Offset	5	0.5	20	±20	191
7	GainX5_Offset	5	0.5	8	±20	120
8	GainX10_Offset	5	0.5	4	±20	106
9	GainX1_Offset	5	1	20	±10	162
10	GainX2_Offset	5	1	10	±10	81
11	GainX5_Offset	5	1	4	±10	33
12	GainX10_Offset	5	1	2	±10	18
13	GainX20_Offset	5	0.125	1	±10	11
14	GainX1_Offset	5	0.125	160	±80	1307
15	GainX2_Offset	5	0.125	80	±80	678

Thermometer Channels Range Definitions

This section provides a description of thermometer channels range definitions.

Table 8-2. Thermometer Channels Range Definitions

Idx	Sensor Measurement Mode
0	DISABLED
1	RTD_2WIRE_50000_OHM
2	RTD_2WIRE_20000_OHM
3	RTD_2WIRE_10000_OHM

Table 8-2. Thermometer Channels Range Definitions (cont.)

Idx	Sensor Measurement Mode
4	RTD_2WIRE_5000_OHM
5	RTD_2WIRE_2000_OHM
6	RTD_2WIRE_1000_OHM
7	RTD_2WIRE_500_OHM
8	RTD_2WIRE_200_OHM
9	RTD_2WIRE_100_OHM
10	RTD_2WIRE_50_OHM
11	RTD_2WIRE_20_OHM
12	RTD_2WIRE_10_OHM
13	RTD_3WIRE_50000_OHM
14	RTD_3WIRE_20000_OHM
15	RTD_3WIRE_10000_OHM
16	RTD_3WIRE_5000_OHM
17	RTD_3WIRE_2000_OHM
18	RTD_3WIRE_1000_OHM
19	RTD_3WIRE_500_OHM
20	RTD_3WIRE_200_OHM
21	RTD_3WIRE_100_OHM
22	RTD_3WIRE_50_OHM
23	RTD_3WIRE_20_OHM
24	RTD_3WIRE_10_OHM
25	RTD_4WIRE_50000_OHM
26	RTD_4WIRE_20000_OHM
27	RTD_4WIRE_10000_OHM
28	RTD_4WIRE_5000_OHM
29	RTD_4WIRE_2000_OHM
30	RTD_4WIRE_1000_OHM
31	RTD_4WIRE_500_OHM
32	RTD_4WIRE_200_OHM
33	RTD_4WIRE_100_OHM

Table 8-2. Thermometer Channels Range Definitions (cont.)

Idx	Sensor Measurement Mode
34	RTD_4WIRE_50_OHM
35	RTD_4WIRE_20_OHM
36	RTD_4WIRE_10_OHM
37	THERMOCOUPLE_J
38	THERMOCOUPLE_K
39	THERMOCOUPLE_T

Thermometer Channels Sample Per Sec Definitions

This section provides a description of the thermometer channels sample per sec definitions.

Table 8-3. Thermometer Channels Sample Per Sec Definitions

Idx	Sample/sec (SPS)
0	2.5
1	5
2	10
3	16.6
4	20
5	50
6	60
7	100
8	200
9	400
10	800
11	1000
12	2000
13	4000

Application Card Hardware Flags

This section provides a description of the application card hardware flags.

- Generic error for all types of application cards:
  - Internal Appcard Error (CODE)
- MS401:
  - OK
  - Open Circuit: the MS401 card's sensor source is in open circuit
  - Overvoltage: the MS401 card's internal voltage protection has been activated
- TH800:
  - OK
  - Sensor Error: a sensor is either unplugged, or the wrong sensor type was set on the TH800 card (works only with PTX type sensors)

Measurement Example

This section provides a description of a measurement example.

Single Diode Measurement (Python)

This measurement is a simple diode measurement written in Python3 language. Before using this sample measurement, you must install the Python3 environment with the 'websocket-client' package. The 'websocket-client' package is used for websocket communication.

For the installation command, see the header of the code snippet:

```
#!/usr/bin/env python3

# Following modules must be installed:
# py -m pip install websocket-client
```

The Hardware setup is the following:

- MS401 card in Slot3: Ch1 is used for a sensor current source (10mA) and for thermal transient measurement.
- LP220 card in Slot1: Ch1 is used for a drive current source (100mA).

The Python script will print its state to the standard output during measurement, and the thermal transient measurement will be saved in the same folder in which the script runs. The script will save the files in both legacy and in the new file format (*PARX*).

You can download the sample script using the following link (using your T3STER's IP address):

http://<ip_address_of_your_t3ster>:8085/assets/docs/meas_examples/diode_transient_example.py