

## **Explanatory Comments to the State Machine version 07 and the Advanced Command Set**

### **Intro**

The GUS standard Interface up to now (version 1.0) has been developed for the “simple” automation of a combined test (Climatic Chamber and Shaker). How an application at the higher level would take over the control and supervision has not been an issue, up to now.

The defined statuses of a test sequence are mainly tailored to the test itself. The original idea was that the supervisory program would not take control of the test. A test is controlled by the control system of the devices at the lower level, while the supervisory program controls the devices. For this reason some statuses (9: device closed and 0: device open) are missing in the first release of the GUS standard interface (version 1.0). These statuses are needed to control and to supervise the devices.

From the other side, some statuses sometimes are artificial, while every device functions differently. In this way the statuses w.r.t. the test sequence eventually do not match the statuses of a device, because the respective device simply does have a specific status. Nevertheless the statuses and their transitions are very important to control a device at any time and to identify potential problems in the setup, the synchronization and the execution of a test. In addition to the statuses of the devices, the polling of certain parameters or variables of the devices is mandatory to be able to synchronize more advanced combined tests. Version 2.0 of the GUS Standard Interface proposes an advanced command set to make the communication about the parameters and variables of the devices possible.

### **Overview of the STATUSes of a DEVICE**

#### **Status 9: closed**

For the supervisory program it is important to know if a device is available or not. It is important to know if the communication with the controller of a device works or not. The command “GUS\_Open\_App” will load the driver. The parameter of the command is the name of the driver, as registered by the Windows® operating system.

The response must be a string: “ACK: device serial/version number”. The device that communicates by means of the selected driver is in the state 9 (closed). The communication with the device has not been set up yet. The status value is 9 and the device is not available yet.

The App is closed again by means of the “GUS\_CloseApp” command. For the “GUS\_CloseApp” command no response from the device is expected.

#### **Status 0: open**

By means of the command “GUS\_OpenDevice” the communication with the device is established. When eventually the communication software is able to communicate with many devices (e.g. S!MPATI communicates with many climatic chambers) the device number has to be given as a parameter of the command (see definition of the GUS standard). Hence the device is identified and the one-to-one communication is established. The status changes to 0 (when the device is available and can be used for the test). After the positive confirmation (“ACK”) and the status changed from 9 (closed) to 0 (open), the device is now available for the test.

When the communication with the device cannot be established, then the status remains 9. The device does not become available for the test.

**Special case:** Today the discussion is still ongoing how a device would react on “GUS\_OpenDevice” when that device already (or still) runs a test. Possibly the device is running our test and the communication has been broken. Then it would be necessary to re-establish the communication again. How the communication can be re-established safely again is still an open issue. On the other hand, when the device is busy with another test, the supervisory program should not intervene with that test and certainly should not stop the running test.

With the command “GUS\_CloseDevice” the communication is stopped. The connection with device does not exist anymore. The status changes from 0 (open) to 9 (close). The device is not available anymore.

### **Status 1: ready**

With the command “GUS\_PrepTest” the test file (or the directory) of the respective device is loaded. Some devices do not open the test file; instead they require the information in which directory the test definition and the test parameters are stored. The status changes from 0 (open) to 1 (ready): the device is available and the test specifications have been loaded. The device is operational and is ready to start the test. The device responds with the string “ACK”.

In the event the test cannot be loaded (the test file or the test directory do not exist or cannot be found) the response must be the string “ERR”. The state remains 0 (open). The device is not ready to start the test.

The command “GUS\_StartTest” starts the test and the status of the device changes from 1 (ready) to 2 (pre-test running). Several devices do not perform a pre-test when a test starts. For that reason it is possible – and allowed – that the status changes immediately from 1 (ready) to 3 (running).

The command “GUS\_CloseTest” closes the test file and the status changes from 1 (ready) to 0 (open). The test is not loaded anymore, but the device remains available and the communication with the device is still established.

### **Status 2: Pre-Test running**

When the pre-test successfully has been finished the status changes from 2 (pre-test running) to 3 (running).

When the device is in the status 2 (pre-test running) also the command “GUS\_StopTest” can be given. In this way the device can be stopped at any time (e.g. emergency stop). The command “GUS\_StopTest” brings the device back into the status 1 (ready). The test file is still loaded.

As already mentioned, not all devices make a pre-test or self-check when a test starts. Eventually a device never reaches the status 2 (pre-test running) and changes to the status 3 (running) immediately.

### **Status 3: running**

After the device reached the status 1 (ready) and it received the command “GUS\_StartTest” the test starts and the device reaches the status 3 (running). By means of the command “GUS\_StopTest” the test can be stopped. The status changes to 1 (ready). The test can be started again.

A test can be paused and can be continued by means of the commands “GUS\_PauseTest” and “GUS\_ContinueTest” respectively.

#### **Status 4: finished**

When the test finishes without disruption the status changes to 4 (finished). A device only reaches the status 4 when the test successfully could be run completely.

**Note:** when a test is running and a failure occurs, the device goes into the state 5 (pause). That gives the operator the opportunity to fix the problem and to continue the test, or to stop the test.

At this point we have to make a distinction between a failure and an error. A failure is considered to be a malfunction that eventually can be solved by the operator. An error typically is a machine malfunction where e.g. the interlock tripped, indicating a more serious problem or an emergency stop that cannot be fixed quickly by the operator.

In the status 4 (finished) the test file is still loaded. The command “GUS\_CloseTest” closes the test file and brings the device back into the status 0 (open). A new test can be prepared.

The command “GUS\_StopTest” brings the device back into the status 1 (ready). The test can be started again.

#### **Status 5: pause**

After the command “GUS\_PauseTest” the device reaches the state 5 (pause). Sometimes, also in this state, the test has to be stopped or has to be interrupted. The command “GUS\_StopTest” brings the device back into the state 1 (ready). The test is still loaded. The test can be started again, or can be stopped as described above.

During the state 5 (pause) the test time and further test parameters will not be reset. The test can be continued by means of the command “GUS\_ContinueTest”. Then e.g. the test time will continue to be updated. During the “pause” time the elapsed time still continues to count.

The command “GUS\_StopTest” will stop the test and bring the device back to the state 1 (ready).

#### **Status -1: error**

In every state a problem can occur. When the device is in the state 1, 2, 3, 4 or 5 and a device failure (e.g. hardware malfunction) occurs, then the state changes to -1 (error). So the state -1 really relates to the malfunction of the device. The test is still loaded. The command “GUS\_CloseTest” closes the test file and the device turns into the state 0 (open). The communication with the device is still running but the test is not loaded anymore.

The “error” state typically is due to a hardware problem (broken power module of the power amplifier / Emergency stop / Interlock tripped / Max. temperature of the climatic chamber exceeded / ...).

### **“GUS\_StopTest” command**

This command brings the device back to the state 1 (ready). This command is available when the device has the state 2 (pre-test running), 3 (running), 4 (finished) or 5 (pause). The test remains still loaded.

### **“GUS\_CloseTest” command**

This command brings the device back to the state 0 (open). This command is available when the device has the state 1 (ready), 4 (finished) or -1 (error). The test file will be closed.

### **Command not matching a state**

In the event a command to a device is given, different from one of the commands for a given state as described in the document “State Machine”, the state of the device will not change.

e.g. when the device is in the 1 (ready) state and the command “GUS\_PauseTest” is given, the device must respond with an error: “ERR” instead of “ACK”. The device remains in the actual state 1 (ready). The actual implementation of the higher level control program will read every “ACK” string as a positive acknowledgment of the given command. Any other string will be recognized as being an error, meaning the given command was not valid for the actual state of the device. The device will not change its actual state.

### **Command Acknowledgement**

After the successful operation, every command has to be confirmed with the string “ACK”, unless defined otherwise. When the operation cannot be completed successfully, the response of the device must be the string “ERR”.

### **Minimum Command Set**

For the automatic start and stop operation by means of a supervisory program, including the supervision of the devices during the test, following commands define the minimum command set that have to be implemented into the GUS standard interface of the device:

- GUS\_Open\_App
- GUS\_CloseApp
- GUS\_OpenDevice
- GUS\_CloseDevice
- GUS\_PrepareTest
- GUS\_StartTest
- GUS\_StopTest
- GUS\_PauseTest
- GUS\_ContinueTest
- GUS\_CloseTest
- GUS\_GetStatus

Further details are described in the appendix II.

By means of the minimum command set, a supervisory program can establish the communication with a device, let the device load its test file, start the test etc. When the communication with several devices has been established, the supervisory program can start the tests of these devices (shaker, climatic chamber, control equipment, etc.), stop or pause the test of a device, continue a test or just stop all tests when an error of a device has been reported. It is the minimum requirement to run a combined test without supervision of an operator.

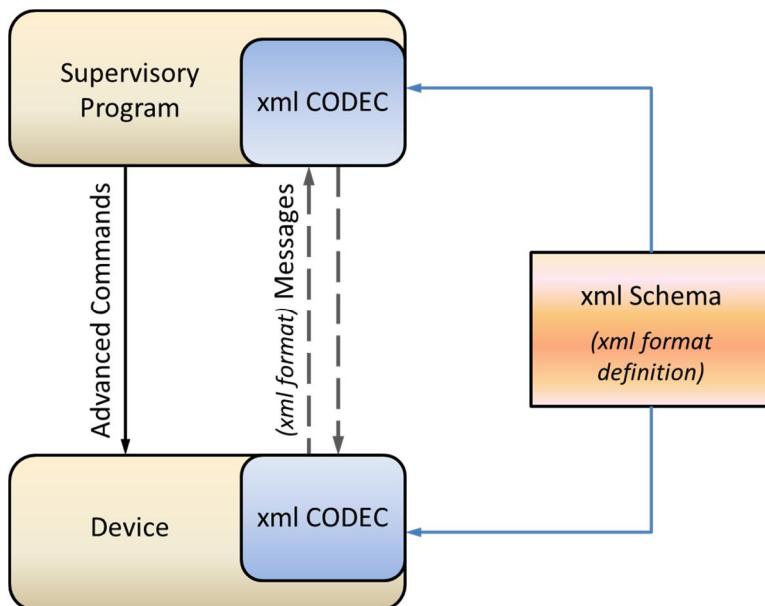
### **Advanced Command Set (*proposed status for version 2.0, not confirmed as "standard" yet*)**

When the supervisory program must be able to get the value of certain device parameters or variables, or when the synchronization of a test requires more detailed information from the devices, the advanced commands must be implemented.

The advanced command “GUS\_SetParameter” even allows the supervisory program to set a certain value in the device, when the respective device parameter (e.g. a setpoint) has been declared available and modifiable by the device.

- GUS\_GetDeviceInfo
- GUS\_GetInfo
- GUS\_GetParameter
- GUS\_SetParameter

Through these commands, the information shared between the supervisory program and the devices, is more complex and requires a structure that defines the format and the type of information, EU (engineering unit), limits, restrictions, etc.



Therefore the communication of the commands listed above is structured into an xml format. XML (Extendible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is defined by the W3C's XML 1.0 specification and by several other related specifications, all of which are free open standards.

An xml schema defines the rules, how the xml language of the messages between the supervisory program and the devices is structured. The xml schema is used by both the supervisory program and

a device to Code and Decode (hence the name “CODEC”) the xml formatted stream of data or messages. The CODEC has to fulfill the rules of the xml schema.

Note: an xml file which is a schema has the extension “.xsd”. The proposed schema for the advanced command set is found in the file “GusDeviceInfo.xsd”. See also appendix IV for detailed information on the schema.

*Principle of operation:*

- a. The supervisory program sends the command “GUS\_GetDeviceInfo” to the device. The device responds with an xml string that defines the information which can be shared between the supervisory program and the device. The contents is a structured list of names of the variables and device parameters, their respective restrictions, limitations, engineering unites, if these device parameters or variables or read-only or not, etc.
- b. With the command “GUS\_GetInfo” the device responds with the actual values of all device parameters and variables that had been declared before as a response to the “GUS\_GetDeviceInfo” command.
- c. The command “GUS\_GetParameter” defines uniquely the device parameter or the variable for which the device has to send back the actual value. For more detailed information, see appendix V.
- d. The command “GUS\_SetParameter” defines uniquely the device parameter or variable and its value which has to be set by the device. The command “GUS\_SetParameter” allows the supervisory program to take control of the device properties: to define a new setpoint, to open or close a relay, to set the value of an analog output, etc. For more detailed information, see appendix V.

*XML format:*

References: <https://en.wikipedia.org/wiki/XML> and [http://www.w3schools.com/xml/xml\\_whatis.asp](http://www.w3schools.com/xml/xml_whatis.asp)

The definitions above will be followed for the implementation of the advanced command set. For more details on the xml format: see appendix III.

The appendix IV gives a detailed description of the “Definition of the Parameters in the xml files of the advanced command set”.

A detailed description of the advanced command set is given in the appendix V. It describes the format of the xml file to exchange the information between a device and the supervisory program.

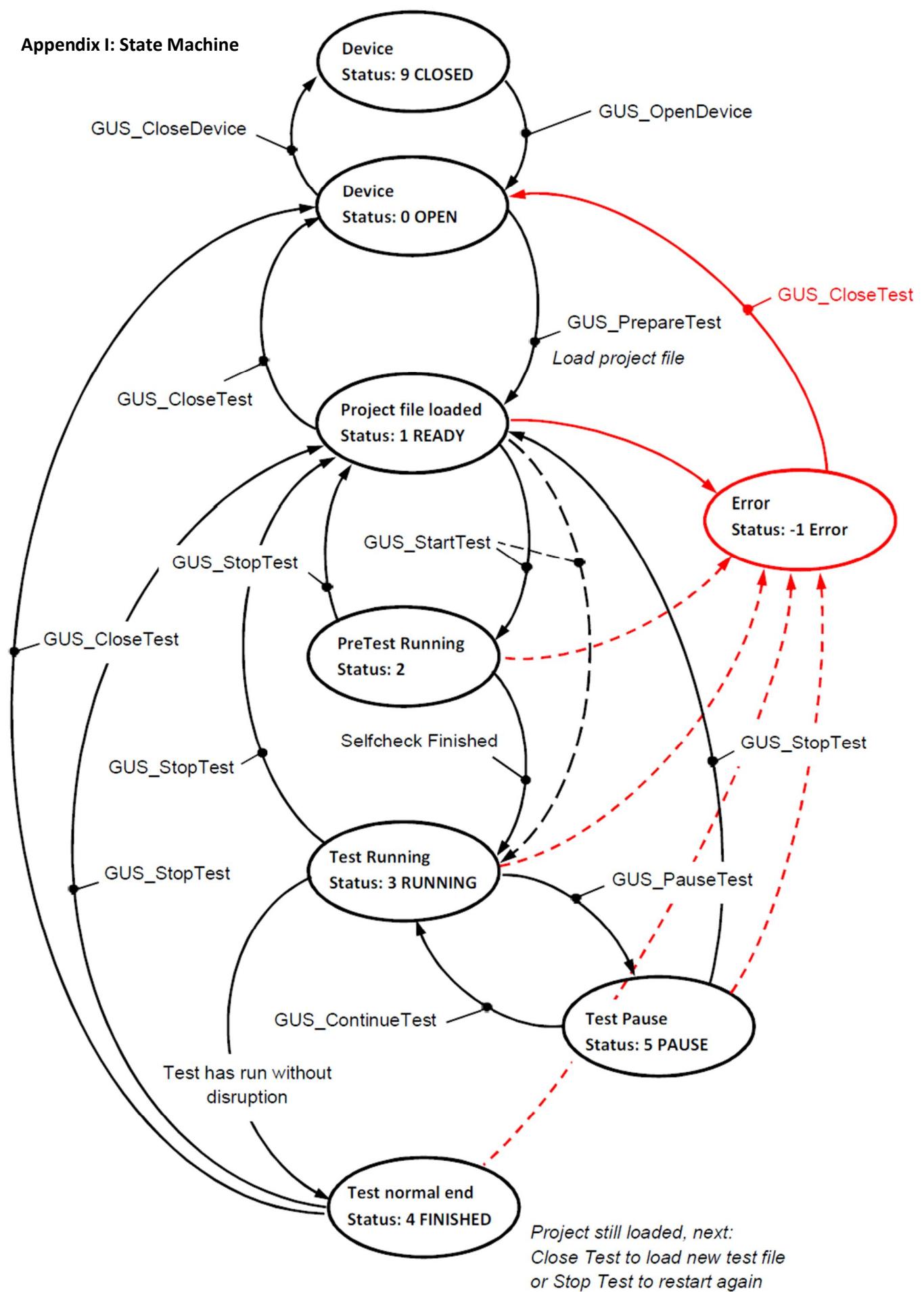
### **Time and Date Format**

For the coding and the representation of the time and date in the xml files, the standard ISO 8601 will be followed.

More details can be found on [https://en.wikipedia.org/wiki/ISO\\_8601](https://en.wikipedia.org/wiki/ISO_8601)

An overview of the general principals is given in the appendix VI.

## Appendix I: State Machine



Overview of the allowed transitions: “state from” to “state to”

State to	9	0	1	2	3	4	5	6	-1
	closed	open	ready	pre-test running	running	finished	pause	busy	error
<b>state from</b>									
9 closed	X	1							
0 open	2	X	3						
1 ready		8	X	4	4				ERR
2 pre-test running			5	X					ERR
3 running			5		X	END	6		ERR
4 finished		8	5			X			ERR
5 pause			5		7		X		ERR
6 busy								X	
-1 error		8							X

## Appendix II: Minimum Set of Commands

Command	Parameter included in the command	Response Format	Response (success)	Response (Fail)
GUS_Open_App	Driver name	string	ACK	ERR
GUS_CloseApp			none	
GUS_OpenDevice	Device ID (number)	string	ACK	ERR
GUS_CloseDevice		string	ACK	ERR
GUS_PreparesTest		string	ACK	ERR
GUS_StartTest		string	ACK	ERR
GUS_StopTest		string	ACK	ERR
GUS_PauseTest		string	ACK	ERR
GUS_ContinueTest		string	ACK	ERR
GUS_CloseTest		string	ACK	ERR
GUS_GetStatus		string	State Number(*)	ERR

(\*) according to the state machine.

## **Appendix III: xml format**

XML documents consist entirely of characters from the Unicode repertoire.

For the list of valid characters: consult page [https://en.wikipedia.org/wiki/Valid\\_characters\\_in\\_XML](https://en.wikipedia.org/wiki/Valid_characters_in_XML)

### **Well-formed Documents**

The XML specification defines an XML document as a well-formed text – meaning that it satisfies a list of syntax rules provided in the specification. Some key points in the fairly lengthy list include:

- The document contains only properly encoded legal Unicode characters
- None of the special syntax characters such as < and & appear except when performing their markup-delineation roles
- The begin, end, and empty-element tags that delimit the elements are correctly nested, with none missing and none overlapping
- The element tags are case-sensitive; the beginning and end tags must match exactly.
- Tag names cannot contain any of the characters !"#\$%&'()\*+;/<=>?@[\\]^`{|}~, nor a space character, and cannot start with -, ., or a numeric digit.
- A single "root" element contains all the other elements.

### **Escaping**

XML provides escape facilities for including characters that are problematic to include directly.

For example:

- The characters "<" and "&" are key syntax markers and may *never* appear in content outside a CDATA section.
- Some character encodings support only a subset of Unicode. For example, it is legal to encode an XML document in ASCII, but ASCII lacks code points for Unicode characters such as "é".
- It might not be possible to type the character on the author's machine.
- Some characters have glyphs that cannot be visually distinguished from other characters: examples are
  - non-breaking space (&#xa0;) " "
  - compare space (&#x20;) " "
  - Cyrillic Capital Letter A (&#x410;) "A"
  - compare Latin Capital Letter A (&#x41;) "A"

There are five predefined entities:

- &lt; represents: <
- &gt; represents: >
- &amp; represents: &
- &apos; represents: '
- &quot; represents: "

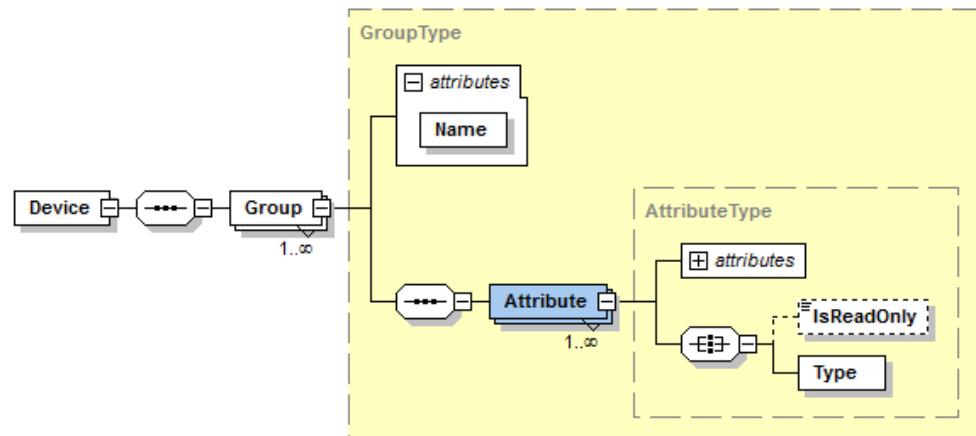
All permitted Unicode characters may be represented with a numeric character reference. Consider the Chinese character "中", whose numeric code in Unicode is hexadecimal 4E2D, or decimal 20013. A user whose keyboard offers no method for entering this character could still insert it in an XML document encoded either as "中" or "&#x4e2d;". Similarly, the string "I <3 Jörg" could be encoded for inclusion in an XML document as "I &lt;3 J&#xF6;rg".

"�" is not permitted, however, because the null character is one of the control characters excluded from XML, even when using a numeric character reference. An alternative encoding mechanism such as Base64 is needed to represent such characters.

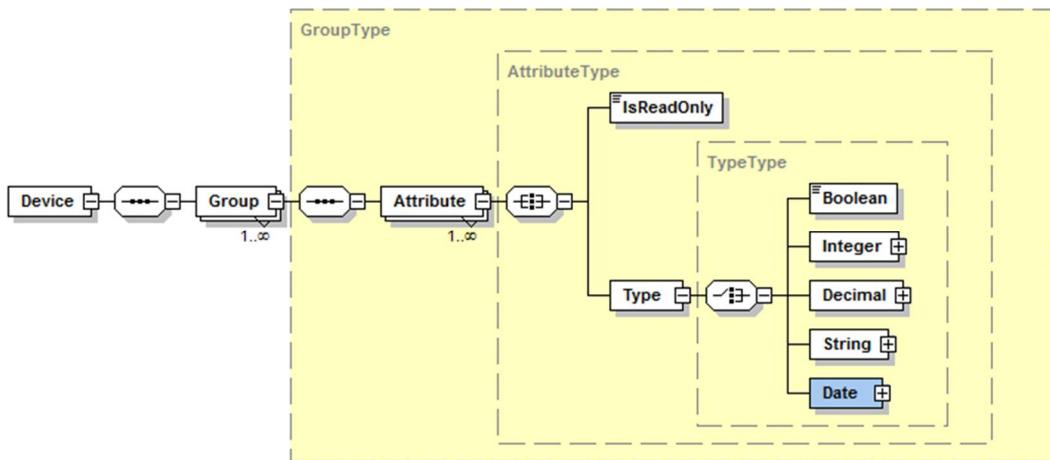
#### Appendix IV: Definition of the Parameters in the xml schema of the advanced command set

Reference: the file GusDeviceInfo.xsd

Following the proposal for the GUS standard version 2.0, the variables and the parameters that define a device are put together in groups:



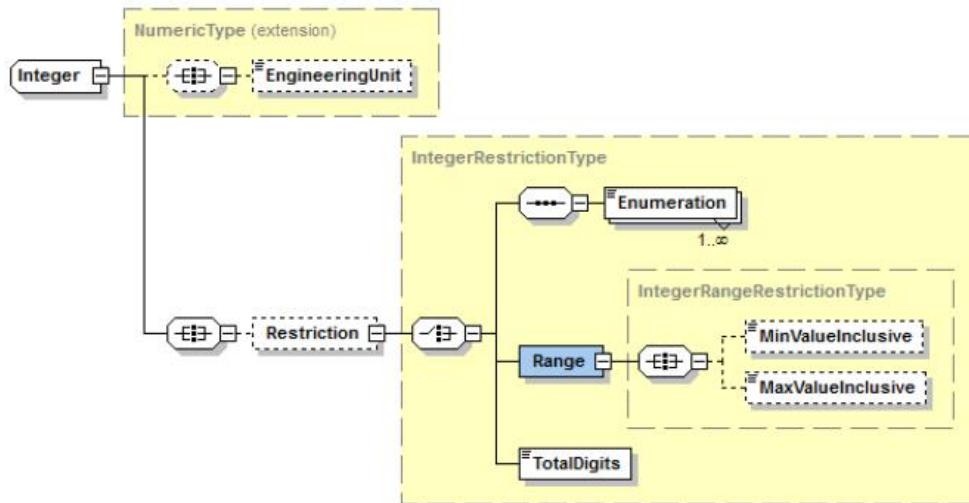
The number of groups is not limited. Typically the first group contains the general information about the device like its name, manufacturer, type or model, etc. The second group contains the control parameters and variables (like the setpoint, actual value, etc.), the third group contains the additional I/O's and so on. Each group must have a Name attribute. The value of this attribute (= name of the group) should be unique for all Groups defined in the xml. Groups are used for logically grouping Attributes (= device parameters or variables) and for 'namespacing' purposes (uniquely identifying an Attribute with the same Name).



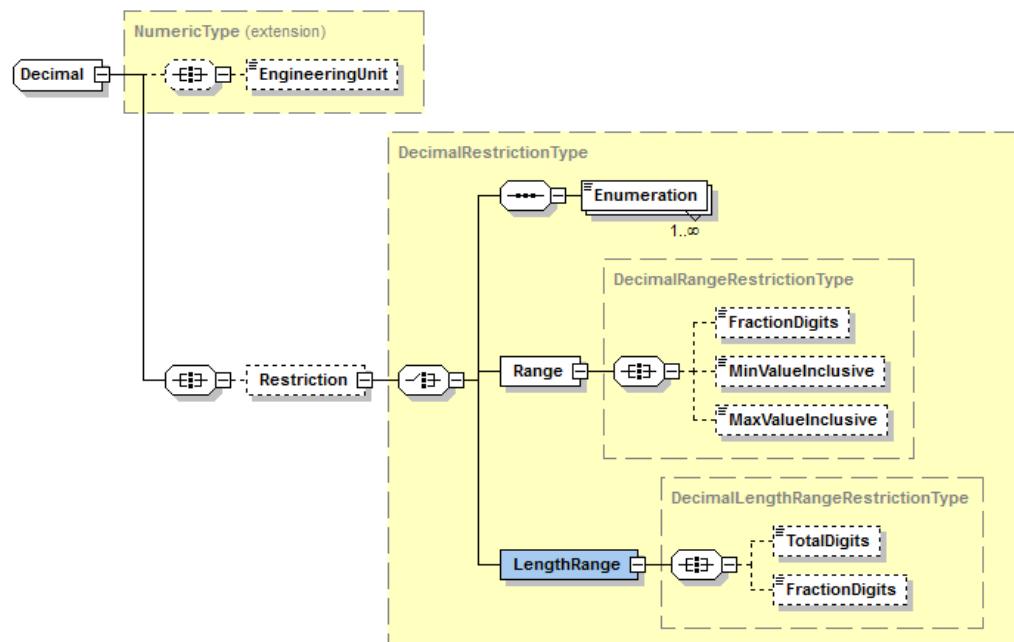
Groups contain one or more Attributes. Also each Attribute must have a Name attribute. The value of this attribute should be unique for all Attributes defined in the Group it belongs to. The control program addresses the Attribute (= device parameter or variable) by its Name (combined with the Group Name). An Attribute can be defined as read-only or not, specifying if its value can be changed by the supervisory program. For some examples: see appendix V, “GUS\_GetParameter” and “GUS\_SetParameter”.

An Attribute has an element called Type of which the type can be: Boolean, Integer, Decimal, String or Date. Each type will be discussed in this section.

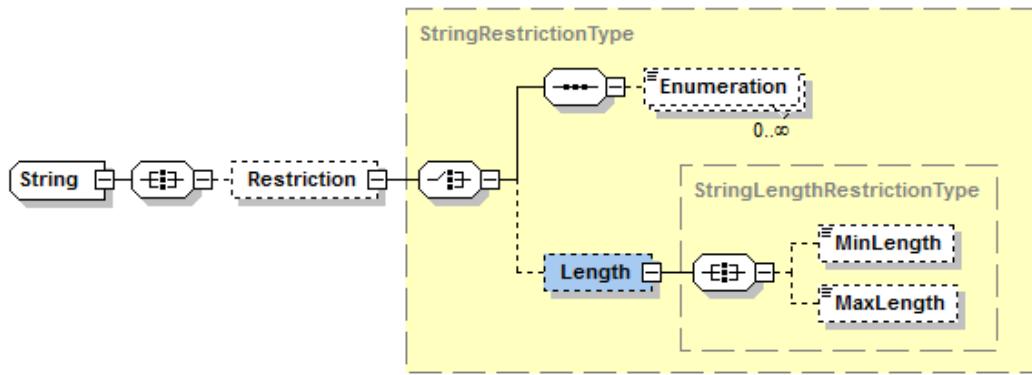
The **Boolean type** only can be 0 or 1, false or true.



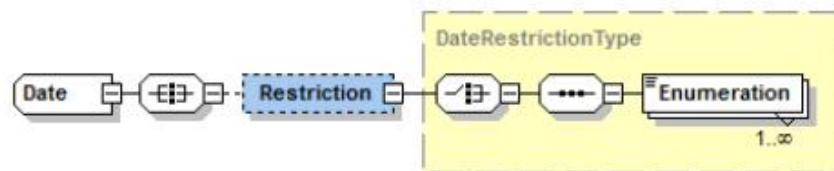
The **Integer type**, next to its restrictions like the minimum and maximum value, total number of digits and eventually a restricted number of values, also can be assigned an EU (engineering unit).



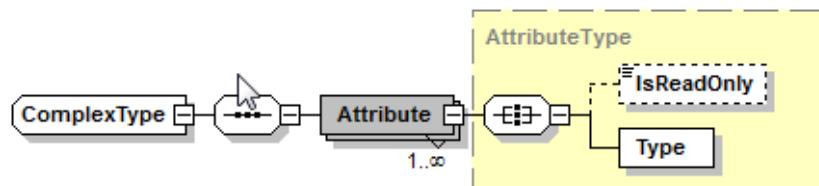
The definition of a **Decimal type** is similar to the integer type, except for the fact that also the number of fractional digits can be defined. For many applications it makes sense to restrict the number of fractional digits to 1 (temperature, humidity, gRMS, ...).



The definition of a **String** is straight forward. Sometimes it makes sense to limit the possible contents of the communicated messages by a predefined set of text strings. Therefore the use of the enumeration makes sense.



The formatting of the **Date** is described in the appendix IV. The standard as described in the ISO 8601 is followed. Also the value of the date can be restricted.



Finally the type of an Attribute can be a **ComplexType**: the purpose of this type is to make nested attributes. The value of the Name of each (nested) attribute should be unique for all Attributes defined in the ComplexType it belongs to.

## Appendix V: Preliminary Information about the advanced command set

### *GUS\_GetDeviceInfo*

---

When the supervisory program sends this command to a device, the device responds with an xml file that contains the definition of all available parameters of the device. In the proposed scheme of the xml files for both the shaker and the climatic chamber following groups are defined:

- DeviceInfo: contains the general information about the device. Name, manufacturer, type, model, serial number and remark.
- ControlledValues: setpoint and actual value controlled by the device (gRMS, temperature, humidity,...). A setpoint can be set by the supervisory program or can be read-only.
- Measurements: additional I/O's available in the device. These I/O's can be analog or digital, can be set or can be read-only, used for additional control or measurements.
- Operation: these device parameters specify which test is running (e.g. sine, random etc. for the shaker system and temperature, humidity, etc. for the climatic chamber).
- Message: the messages defined in this group add additional information to the general communication between the supervisory program and the device: alerts, alarms, etc.
- Testing: the set of device parameters add information to the test running by the device: elapsed time, remaining time, step number in the test program, repeats, etc.

Detailed examples can be found in the files “**Gus\_Device\_Shaker\_V01.xml**” and “**Gus\_Device\_Chamber\_V01.xml**” as an example for the shaker system and the climatic chamber respectively.

It must be noted here, that the list of information can be shortened or extended. The xml file structure is flexible to add new parameters that are not defined in the examples, or to leave out parameters that are not available.

When the device responded with this xml file, the supervisory program knows the definition of all parameters that are available and can be sent or received by the device. The xml file defines the format, eventual restrictions, engineering unit, etc. for each parameter.

### *GUS\_GetInfo*

---

The command demands the actual values of the parameters available from the device. The device responds to the supervisory program with an xml file, structured in the same way as described above (*GUS\_GetDeviceInfo*) but this time only with the list of parameters and their respective values. Of course, the values must adhere to the formats set forward in the xml file, sent as the response to the “*GUS\_GetDeviceInfo*” command.

### *GUS\_GetParameter*

---

Instead of requesting all information at once with the command “*GUS\_GetInfo*” in many cases it makes sense to ask for the value of one specific device parameter or variable. The command “*GUS\_GetParameter*” allows the supervisory program to ask the device to report one specific value.

The parameter of the command (= call by the supervisory program) is a string that defines uniquely the device parameter or variable for which the actual value is requested. The format of the string is in the xml format. The contents, name and structure, has been defined by the device in the response to the command “GUS\_GetDeviceInfo”.

As an example to ask for the actual (controlled) temperature of the climatic chamber:

```
<Device>
  <ControlledValues>
    <Temperature>
      <CurrentValue></CurrentValue>
    </Temperature>
  </ControlledValues>
</Device>
```

E.g. In the example above, the parameter (= string) of the command “GUS\_GetParameter” sent to the climatic chamber to get the actual value of the temperature should look like:

```
<Device><ControlledValues><Temperature><CurrentValue></CurrentValue></Temperature></ControlledValues></Device>
```

The device answers with the same structure to identify the parameter, but this time including the actual value of the parameter (in the example 101.4 °C):

```
<Device><ControlledValues><Temperature><CurrentValue>101.4</CurrentValue></Temperature></ControlledValues></Device>
```

When the device parameter or variable does not exist, the device must respond with “ERR”.

### GUS\_SetParameter

---

The value of some parameters can be set by the supervisory program. This is only possible and allowed when the respective property of the parameter is set correctly:

```
<IsReadOnly>false</IsReadOnly>
```

Taking the example from above, the command from the supervisory program could be:

```
<Device>
  <ControlledValues>
    <Temperature>
      <DemandValue>150.0</DemandValue>
    </Temperature>
  </ControlledValues>
</Device>
```

Then the parameter string of the command “GUS\_SetParameter” must be:

```
<Device><ControlledValues><Temperature><DemandValue>150.0</DemandValue></Temperature>
</ControlledValues></Device>
```

This command sets the setpoint of the temperature control in the climatic chamber to +150.0°C.  
When the device parameter or variable does not exist or is read-only, the device must respond with  
“ERR”.

## **Appendix VI: Time and Date Format**

### **ISO 8601 General Principal**

- Date and time values are ordered from the largest to the smallest unit of time: year, month, (eventually: week), day, hour, minute, second.
- Each date and time value has a fixed number of digits that must be padded with leading zeros.
- Representations can be done in one of two formats – a basic format with a minimal number of separators or an extended format with separators added to enhance human readability.
- For reduced accuracy any number of values may be dropped from any of the date and time representations, but in the order from the least to the most significant.
- If necessary for a particular application, the standard supports the addition of a decimal fraction to the smallest time value in the representation.
- Weeks are numbered from 01 to 53 and are preceded with “W”.

### **Examples**

#### ***Combined date and time:***

YYYYMMDDThhmmss.sss or YYYY-MM-DDThh:mm:ss.sss

YYYYMMDDThhmm or YYYY-MM-DDThh:mm

#### ***Example date:***

YYYYMMDD or YYYY-MM-DD

*Exception:* YYYYMM is not allowed. Must always be: YYYY-MM

YYYYWxx or YYYY-Wxx (note: xx = week number 01 through 53)

YYYYWxxD or YYYY-Wxx-D ( note: D = weekday numbered 1 through 7)

#### ***Example time:***

hhmm or hh:mm

hhmmss or hh:mm:ss