Programmer's Guide

ZES ZIMMER Electronic Systems GmbH
Tabaksmühlenweg 30
D-61440 Oberursel (Taunus), FRG

Phone +49 (0)6171 628750
Fax +49 (0)6171 52086

E-mail: sales@zes.com
Internet: www.zes.com

Regard DIN 34!

We reserve the right to implement technical changes at any time, particularly where these changes will improve the performance of the instrument.

Content

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

Figures

Examples

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

1 Introduction

This programmer's guide was written to clarify some important points which occur when programming measuring instruments. It was written especially for the current series of precision power meters from ZES ZIMMER Electronic Systems (LMG95, LMG450 and LMG500). Due to the compatibility of our instruments to standards like SCPI, this guide is also a general purpose help when programming instruments from other manufacturers.
This guide offers

- A short introduction into the programming. We start with a simple example to show what to do. This example has a lot of links to the detailed descriptions. See chapter 1, 'Introduction'
- A detailed description of all steps, from a detailed syntax description to the handling of the interface. See chapter 2, 'Description'
- Many hints for efficient programming in your application. See chapter 3, 'Advanced programming'

This guide is only available in English. This should be no practical problem, because most literature for programmers is written in English.

Petition
This guide was written on suggestion of some users. We tried to implement the answers on everything which we were asked for. But of course such a guide will never be complete and perfect. So please

- if you have any questions, please contact us (www.zes.com).
- if you have any proposals what to add to or change in this guide, please contact us.
- if you find any errors in this guide, please contact us.

This guide can just be improved with the help of you, the users.
Thank you very much for your help!

1.1 Interface types

To communicate with a computer you need a computer interface in your measuring instrument. Despite modern standards like USB or Ethernet, the RS232 and the IEEE488 interface are still the most commonly used interface types today for high end measuring equipment.

There are some main differences between RS232 and IEEE488:

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

- The costs of an IEEE488 (sometimes also called GPIB = General Purpose Interface Bus) card are high. The RS232 is implemented in most PCs. For some modern PCs which only have USB interfaces, ZES offers USB to RS232 converters. The drivers of these converters create a virtual COM port, so the programming is the same as for a real RS232 interface.
- To an RS232 interface you can connect only one instrument. The IEEE488 can connect up to 30 instruments.
- The maximum data speed of RS232 is 22.5KByte/s, IEEE488 can reach 1MByte/s.

Further information about these interfaces you can find in chapter 2.1, 'Background of RS232, IEEE488 and SCPI'.

1.2 Applications

There are two typical applications, how an instrument is used together with a computer. A single instrument is typical in applications where you have to get and store measuring values for a longer time. Or where you want to get and store measuring values if some set-up is finished and you want to document this. Further on you can set-up an instrument for a measuring application.

For this purpose you can use already existing programs like 'LMG-CONTROL' from ZES. This program configures the instrument and then logs the data as long as you want. This can be done via RS232 or IEEE488 interface. The data can be exported to ASCII format and then be used in other applications (like Excel) for further evaluation.

Such a terminal program can always control just one instrument. If you have more measuring instruments or other instruments (like function generators, programmable loads, frequency converters ...) you will need other solutions.

In other words: If you have more than one instrument to handle you can perhaps use several independent programs. But there are several disadvantages:

- synchronisation problems between the programs
- writing to different files for storing the measuring results, so it is difficult to find corresponding data.
- IEEE488 bus controller cannot be handled by different independent programs, so just one instrument can be used!

The best way in this case is to write your own program. General purpose programs for this case cannot exist.

To give you some help for doing this is the task of this manual.

1.3 Concept

In principle the communication is a game of question and answer. You send a message (details follow in 2.2, 'Program messages') to the instrument. This message is executed

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

(details follow in 2.4, 'Data flow, system structure'). If the message requires an answer, the instrument will send back this answer (details follow in 2.3, 'Response messages'). To send a message you need a communication interface.

There are four important functions for the communication from your PC to the instrument:

- Open the interface (details follow in 2.5, 'Opening the interface')
- Close the interface (details follow in 2.6, 'Closing the interface')
- Write data to the instrument (details follow in 2.7, 'Writing to the interface')
- Read data from the instrument (details follow in 2.8, 'Reading from the interface')

With these basic steps we could start to write our first program.

1.4 Programming language

There are a lot of programming languages. Some support IEEE488 interface, some not, some are easy to handle, some have graphical symbols instead of the typical source text. Frequently used languages are:

- C/C++
- LabVIEW
- LabWindows
- Basic

There are a lot of other possibilities. If you got this manual as file, there was also delivered a 'Python' interpreter with a lot of examples. Python is a language which is easy to learn and to understand. The advantage is, that Python is freeware and you can get and modify/extend the sources. For more details see http://www.python.org

ZES ZIMMER has written a RS232 driver to enable easy access from Python to this interface. That is the reason, why we deliver this program together with this manual. You can work directly with the examples and your instrument and you can modify the examples and test what happens.

Nevertheless those are just simple examples which are easy to understand and to transform in any other language you prefer for programming. You will not have to learn Python to understand this guide.

1.5 Hello world

Like in other programming languages we could output this simple text. But it is no problem to get instead the first real answer from the LMG. The first test program is:

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

```
#
# HELLO.PY
#

# load ZES RS232 functions
import zes

# Open the serial interface:
LMG=zes.Open("COM1", "BAUD=38400 EOS=LF PROTO=RTS/CTS")

# Write command to instrument
zes.Write(LMG, "*IDN?")

# Read answer from instrument
answer=zes.Read(LMG)

# Output answer on screen
print "The connected instrument is:\n"+answer

# Write command to instrument
zes.Write(LMG, "GTL")

# Close connection
zes.Close(LMG)
print "\nProgramm finished"
```

Example 1: Program HELLO.PY

The parts of this program are explained in the following. Comments in Python start with a '#' and are valid until the end of the line.

import zes	Includes the ZES library for RS232 support. Now you can use all commands starting with 'zes.'
LMG=zes.Open("COM1", "BAUD=38400 EOS=LF PROTO=RTS/CTS")	Open the serial interface COM1 at your computer with 38400 baud, EOS is <lf> (linefeed) and protocol is RTS/CTS. This interface is referenced from now with the variable 'LMG'. Please make sure, that you have set-up the LMG with exactly the same data!
zes.Write(LMG, "*IDN?")	Send the command '*IDN?' (Identify) to the instrument with the reference 'LMG'. Thereby the identification of the instrument is requested.
answer=zes.Read(LMG)	Read the answer from the instrument and store it in the variable answer.
print answer	Output this variable on the screen.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

zes.Write(LMG, "GTL")	Send the command 'GTL' (Go to local) to the instrument. After this you can operate it again from its panel.
zes.Close(LMG)	Close the interface.

Before you can start this program you have to set-up the communication. Use a simple RS232 cable (1:1, no null modem function) and connect it to COM1 of your PC. Connect the other side to 'COM A' or COM1 of your instrument. If COM1 of your PC is not available you can also use COM2 or any other COM port. In this case you have to change the parameter to the zes.Open line in the program.

Now switch on PC and instrument and set-up the following in the IF/IO menu:
COM A, 38400 Baud, EOS = <lf>, Echo off, Protocol RTS/CTS. Usually you get these values when selecting 'COM A OEM Appl. with 38400 Baud'. See instrument manual for details.

The instrument is now set-up in the same way as the program above will open the communication port. This is very important!

To start this program install the Python directory on your computer. Now change to your DOS prompt and enter:
bin\python hello.py

You could also double click on the icon of hello.py and set-up, that this type of file should be executed by the python.exe file. But please note that the output window might close after executing the program and you will not be able to read the messages.
If you have a working connection to the instrument, the instrument should answer with its identification which will look like this:

ZES ZIMMER Electronic Systems GmbH, LMG95, 04700102, 3.087

With the common command `*IDN?` the instrument is requested to return its identification. According to the standard the ZES ZIMMER instruments return several information in four fields which are separated by comma.

Field 1: Manufacturer
Field 2: Model
Field 3: Serial number
Field 4: Software version

The last three fields depend of course on your individual instrument.

If this example fails please check the following and try again:

- Is the COM port of your PC really available? Sometimes other programs block it.
- Is the hardware of this COM port ok? Check it with any other serial communication or use another PC for this test.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

- Did you use a correct cable? The cable has to have all 9 wires, not just 3! Do not use a null modem when connecting to COM A of the instrument.

If the problems do not vanish, please contact us:
http://www.zes.com

Note
The above program will work under usual conditions. But there are a lot of improvements, starting with the opening of the interface, reading values, ...

These improvements would be confusing at this time and would obscure the important basics, which should be demonstrated. But of course they are explained later on.

2 Description

2.1 Background of RS232, IEEE488 and SCPI

For the data exchange between PC and measuring instrument we use the most common interfaces: RS232 and IEEE488.

The RS232 is a serial interface which is used between one PC and one instrument.
The IEEE488 is a parallel interface between one PC and one or several instruments. The relevant standards are IEEE488.1 and IEEE488.2, extended by SCPI (Standard Commands for Programmable Instruments).

The first developments were done by HP (Hewlett-Packard) under the name GPIB (General Purpose Interface Bus) in the 1970's. It described the physical functions of the interface like signals, wires, hardware protocol, dimensions, ...

Further on some universal commands were defined, which are transferred by special states of the control signals and not on the usual data path. These universal commands were not used for data exchange, but for controlling the instrument (e.g. device clear, remote enable, ...).

This was converted to IEEE488.1 in the mid 80's. Up to then there was no standard, which commands should be sent. Each manufacturer used his own command set. This could be very confusing.

The IEEE488.2 defines very exact rules for the instrument syntax as well as some common commands. These commands are transferred as ASCII commands via the usual data path. The common commands are commands which apply to all instrument (like '*IDN?' to get the instrument identification) and do not take care about special instrument features. Each common command starts with a '*'.

SCPI extends the IEEE488.2 rules by device specific commands for many measuring values. By this it should be possible to connect instruments from different manufactures to a bus system and get the same results with the same commands.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

The SCPI commands are also ASCII strings.
The SCPI commands are arranged in a tree structure with a root (similar to a PCs file system path).

In contrast to this, a universal command (IEEE488.1) can be sent at any time, even while a common command/device specific command is sent or executed. This is very important to keep in mind for one of the following sections.

2.2 Program messages

A message to the instrument can be composed of several commands (common commands and device specific commands, but not universal commands!). It is an ASCII string which is terminated by an EOS (= end of string) symbol.

The commands can be either in SCPI (see chapter 2.2.6) or SHORT (see chapter 2.2.7) language. You can select the language with a special command. Following you see some complete examples from the LMG user manual. They are used to explain the syntax features in the next chapters.

SCPI: :FETCh[:SCALar][:VOLTage]:DC?/qonly/ | :READ[:SCALar][:VOLTage]:DC?/qonly/
SHORT: UDC?/qonly/
ID: Udc

Example 2: Reading the DC voltage

SCPI :MEMory:FREeze <NRi>
SHORT: FRZ <NRi>
ID: no ID defined

Example 3: Freezing scope memory

SCPI: :*RST /nquery/
SHORT: *RST /nquery/
ID: no ID defined

Example 4: Reset of instrument

The following rules will explain all details of the above examples. If not specified otherwise, they are valid for SCPI and SHORT commands.

2.2.1 Identification (ID)

The ID field in the above examples shows, which identifier has to be used in the formula editor of the instrument itself. Values which cannot be used there have no ID. In general this has nothing to do with the programming of the interface. An exception from this rule you can find in chapter 3.4.2, ‘How to change the analogue outputs?’.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

2.2.2 Combining commands

When you combine several commands in one program message, you have to separate them with a semicolon ';'.

Example
You want to combine the SHORT commands of the above 'Example 2: Reading the DC voltage' and 'Example 3: Freezing scope memory'. Then you have to input:
UDC?;FRZ 1

2.2.3 Small and capital letters

It does not matter if you send a command in lower-case or in upper-case letters to the instrument. The execution of commands in capital letters is a little bit faster, but not much. The reason why SCPI commands are written as a mix of small and capital letters is described in 2.2.6.1, 'Longer and shorter SCPI commands'.

2.2.4 Channel number (suffix)

To get the measuring values from any desired measuring channel you have to enter the so-called suffix. If you do not enter a suffix, a value of 1 is assumed. Following you see the 'Example 2: Reading the DC voltage' to explain the handling of the suffix.

Reading DC voltage of channel 1
SCPI: :FETCh:SCALar:VOLTage:DC1?
SHORT: UDC1?

is equal to

SCPI: :FETCh:SCALar:VOLTage:DC?
SHORT: UDC?

Reading DC voltage of channel 2
SCPI: :FETCh:SCALar:VOLTage:DC2?
SHORT: UDC2?

The suffix has to be inserted directly after the command (without any space) and directly before a '?' (if existing, also without any space).

2.2.5 /qonly/, /nquery/ and `?`
If '/qonly/' is added to a command description (see 'Example 2: Reading the DC voltage') it is only possible to request a value (query only). This request is initiated by a '?' after the command or suffix (if specified). This '?' has to follow without any space!

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

A '/nquery/' instead of '/qonly/' indicates, that this command can only be executed. It will not send back any reply (no query, see 'Example 4: Reset of instrument').

The texts '/qonly/' and '/nquery/' should not be sent to the instrument! Commands without '/nquery/' or '/qonly/' (see 'Example 3: Freezing scope memory') can be set and requested.

Example: /qonly/

:FETCh:SCALar:VOLTage:DC?	ok, just reading the voltage
:FETCh:SCALar:VOLTage:DC	fail! You cannot set this value

Example: /nquery/

*RST	ok, force instrument to reset
*RST?	fail, this command cannot return a value

Example: Neither /nquery/ nor /qonly/

:MEMory:FREeze ON	ok, freeze the actual values ('ON' is the NRi parameter)
:MEMory:FREeze?	ok, read the actual freeze state

2.2.6 SCPI language

The SCPI commands are arranged in a tree structure. This represents a general instrument structure like in the following figure:

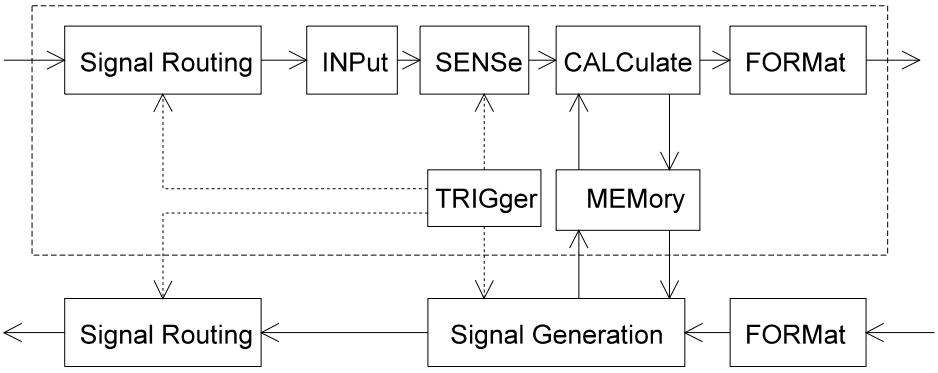

Figure 1: SCPI Functional Model

Beside the structures for measuring in the picture above, the LMG instruments also have some further structures like DISPlay, INSTrument, ...:

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

:CALCulate

Defines how to calculate formulas or limits

:DISPlay

Defines display brightness and contrast

:FETCh

Reads measuring values immediately

:FORMat

Defines the data output format

:INITiate

Defines when to copy or output data

:INPut

Defines the signal coupling

:INSTrument

Defines the measuring mode

:MEMory

Defines the freeze of the scope data buffer

:READ

Reads measuring values after the actual cycle

:SENSe

Defines measuring conditions like ranges, testing times, ...

:SOURce

Defines the outputs of the processing signal interface

:STATus

Defines the behaviour of several internal registers

:SYSTem

Access to error messages, system time & date, ...

:TRIGger

Commands for synchronisation and start/stop of measuring. To explain this structure we use dummy commands instead of the real commands. They are simpler for understanding the principle.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

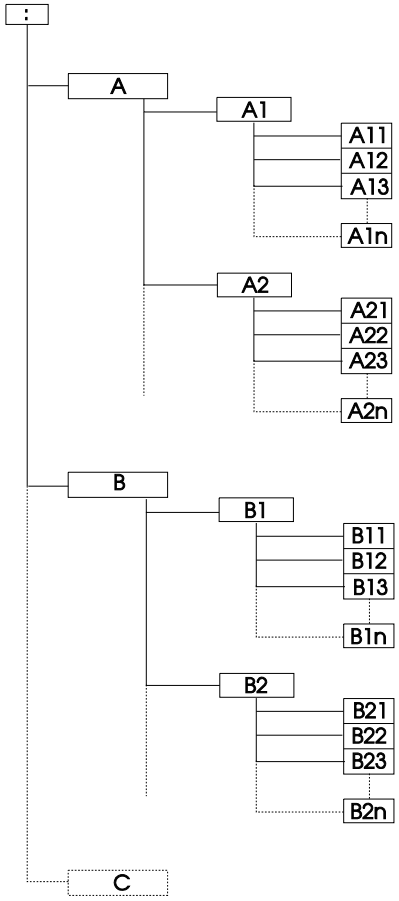

Figure 2: General tree structure

Figure 2 shows how commands can be combined. All commands need a root (similar to the 'C:\' in DOS). In case of SCPI the root is a simple colon ':'. The first command of a transfer does not need the ':' because the first command always starts at the root.
The branches of the tree are separated by a ':' (similar like the '\' in DOS).

Example: Calling command A11

:A:A1:A11 with leading colon

is equal to

A:A1:A11 without leading colon

If there are several commands at one level which should be executed you can combine them (see also 2.2.2, 'Combining commands'). These combined commands share the same prefix until you start again at the root. Due to this you can remove the header for the following commands. To separate the commands you have to use the semicolon`;` as separator.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

Example: Calling commands A11 and A12

A:A1:A11;:A:A1:A12	ok, semicolon separates the commands, second command starts at the root
A:A1:A11;A12	ok, because the commands share the same prefix. Prior to 'A12' the prefix 'A:A1:' from the first command is placed automatically
A:A1:A11;A:A1:A12	fail! The colon after the semicolon is missing. Therefore the prefix 'A:A1:' from the first command is placed in front of the second command which is then read as 'A:A1:A:A1:A12'. This command does not exist!
A:A1:A11; A12	fail! Each new line starts at the root, so the command ':A12' is searched for but not found.

As you can see there are several possibilities to send a command to the instrument.

2.2.6.1 Longer and shorter SCPI commands

There are two possible spellings in the command description for SCPI-commands, a long one which is easier to read and a short one which is shorter and a little bit faster.

The part of the SCPI command written in capital letters is the shorter spelling (this has nothing to do with the SHORT language of the ZES ZIMMER instruments, see 2.2.7, 'ZES SHORT language'). The capital and small letters together are the longer spelling.

The following example shows some of the possibilities (see 'Example 3: Freezing scope memory'). They are written in capital letters, but also work in small letters (see 2.2.3, 'Small and capital letters').

:MEMORY:FREEZE	ok, long spelling
:MEM:FRE	ok, short spelling
:MEMORY:FRE	ok, mix of long/short spelling in different command parts
:MEM:FREEZE	ok, mix of long/short spelling in different command parts
:MEMO:FREEZE	fail! neither long nor short spelling for 'MEMory'.

2.2.6.2 Default values like [...]

The commands in square brackets are default values in the LMG instruments. It is not necessary to enter them, but you can if you like. In 'Example 2: Reading the DC voltage' the commands `:SCALar` and `:VOLTage` are default values. Please note, that the square brackets must not be sent to the instrument. They are only in the description to mark the default values. All the following commands do exactly the same:

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

:FETCh:SCALar:VOLTage:DC?	Including the default values
:FETCh:DC?	Without both default values
:FETCh:SCALar:DC?	Without the second default value
:FETCh:VOLTage:DC?	Without the first default value

2.2.6.3 The `|`-Symbol (or)

The '|' symbol (see 'Example 2: Reading the DC voltage') is equal to a logical 'or' function. It is used in the LMG manuals to separate two commands which request the same value in two different ways. In 'Example 2: Reading the DC voltage' the DC value of the voltage is requested in all cases, in one case by the FETCh and in the other case by the READ command (for differences see chapter 2.8.2; ':FETCh, :READ and SHORT'). To keep the explanations as short as possible these two commands are separated by the symbol '|'. It is not a valid symbol in IEEE488.2 or SCPI and should not be sent to the instrument.

2.2.7 ZES SHORT language

Beside the SCPI commands the LMG instruments from ZES ZIMMER also offer an alternative language, which is unique for ZES products. The main advantage of this SHORT language is that the commands are usually just 4 bytes long. So these commands can be parsed very efficiently and the execution is much faster.

The SHORT commands have a flat structure. The lack of a tree structure is another reason for their faster execution. The LMGs are always starting up with the SCPI command set. To use the SHORT commands you have to change the language with the SCPI command

:SYSTem:LANGuage SHORT

To reach the SCPI language again, please enter the SHORT command

LANG SCPI

The following example shows a SCPI command and its SHORT equivalent.

Reading of DC voltage
:FETCh:SCALar:VOLTage:DC?

is equal to

UDC?

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

Please note
The SCPI commands have a shorter and a longer spelling (see 2.2.6.1, 'Longer and shorter SCPI commands'). None of them is comparable to the SHORT command set. This command set is unique for ZES ZIMMER power meters.

2.2.8 Parameters

For some commands it is possible and/or necessary to add parameters. They are specified in the syntax explanation in angle brackets. In contrast to the suffix or the question mark the parameters have to be separated by a space. They are placed after command, suffix and '?'. There are several types of parameters:

2.2.8.1 <NRi>, <NRf>

<NRi> and <NRf> are symbols for flexible numerical formats. The `i` in <NRi> stands for 'integer', the 'f' in <NRf> for 'float' numbers.

The data format is very flexible, so you can send any usual format: '123', '123.0' or '+1.23e+2' are understood as <NRf> numbers. A complete description of all possible syntax you can find in section 4 (decimal numeric program data) in Figure 9. For <NRi> you find the rules in section 5 (nondecimal numeric program data), section 4 (decimal numeric program data) or section 2 (character program data) in Figure 9.

Following you find three examples which show possible numbers according to the sections 2, 4 and 5. All examples force the instrument to change into the CE measuring mode.

Calling with rules of section 4 (decimal numeric program data)
SCPI: :INSTrument:SELect 2
SHORT: MODE 2

Calling with rules of section 5 (nondecimal numeric program data)

Binary data
SCPI: :INSTrument:SELect #B10
SHORT: MODE #B10

Hex data
SCPI: :INSTrument:SELect #H2
SHORT: MODE #H2

Calling with rules of section 2 (character program data)
SCPI: :INSTrument:SELect CEFLK
SHORT: MODE CEFLK

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

In the last example you could see that not only numbers are allowed, but also strings which are synonymous to numbers. The following table contains all string/number combinations which are implemented into ZES ZIMMER instruments:

String	Off	On
Value	0	1

String	Manual	Auto
Value	0	1

String	INT	EXT
Value	0	1

String	ASCII	PACKED
Value	0	1

String	NORML	CEHRM	CEFLK	HRMHUN	TRANS
Value	0	1	2	3	4

String	SCPI	SHORT
Value	0	1

String	LINE	EXTS	U	I
Value	0	1	2	3

String	ACDC	BP	AM
Value	0	1	2

It does not matter which string follows a command, it only has to have the same value. Example:

:INSTrument:SELect CEFLK

is equal to

:INSTrument:SELect U

or

:INSTrument:SELect AM

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

All these commands change to 'CE Flicker' mode of the instrument. But the second and third version are worse to read and should not be used.

2.2.8.2 <list>

With the <list> parameter you can get several values with one command from a list or array. <list> is a short form for <(<NRf>:<NRf>)>. The results are separated by comma, see following example:

Output of AC voltage and the harmonics of order 1 to 3
You have to send (here as SHORT command):

UAC?;HUAM (1:3)?

The instrument response will be

223.45;221.75,0.3456,5.6789

The first part of the answer is the AC value of the voltage. The following semicolon separates this answer from the second part of the request (the requests were also separated by semicolons!). In the second part of the answer the three values are separated by comma.

2.2.8.3 <string>

This parameter is used to transfer strings. This is enclosed in quotation marks ' " '. Inside a string all characters are allowed, also <cr> ('\r'), <lf> ('\n'), <cr><lf> ('\r\n') and other possible EOS signals (see 2.7.2, 'EOS and EOI'). The quotation mark itself can be transferred by a doubling it ("").

Caution!
If you forget the closing quotation mark all following characters (also commands, EOS, ...) will be interpreted as part of the string. So they are not executed. This can cause the instrument to be blocked. To solve this, you have to reset the interface part of the instrument (see 2.5.2, 'Resetting the LMG interface') or send a closing quotation mark.

Examples
Following text should be transferred:

Peter said „Ok" and vanished

The small pictures are taken from 'Figure 9: Overview Program Data' and explain the special underlined situation.

"Peter said ""Ok"" and vanished" — ok, quotation mark at start and end, two double quotation marks enclosing 'Ok'

"Peter said ""OK"" and vanished"

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

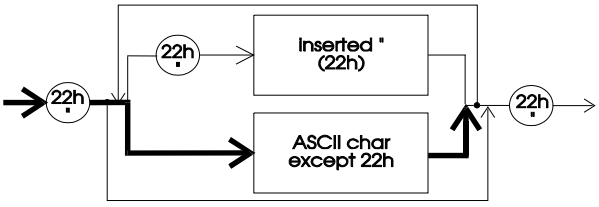

Figure 3: Start of string

“Peter said ““OK““ and vanished“

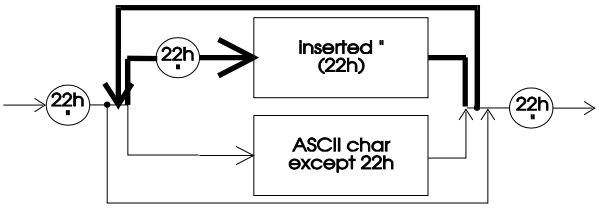

Figure 4: Quotation mark inside a string

“Peter said ““OK““ and vanished“

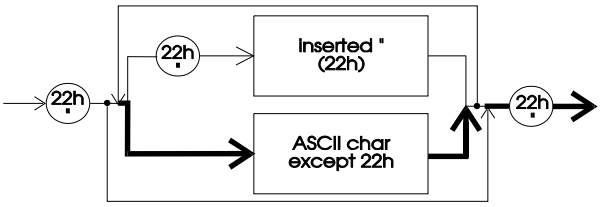

Figure 5: End of string

The following example will not work:

“Peter said “OK“ and vanished“	fail! The quotation mark after ‘said’ terminates the string!

“Peter said “OK“ and vanished“

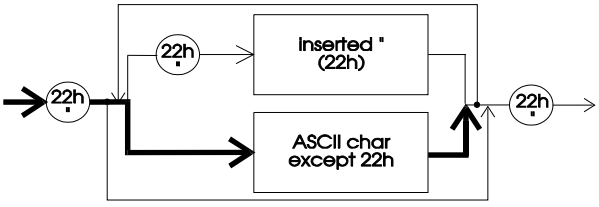

Figure 6: Start of string

“Peter said “OK“ and vanished“

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

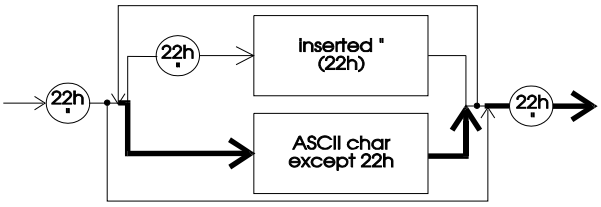

Figure 7: End of string

In contrast to Figure 5 the second (single!) quotation mark terminates the message. The instrument expects a command separator or an EOS after a string, but it gets an 'OK" and vanished"'. So an error message is generated.

Another more practical example can be found in 'Example 5: Program STRING.PY' in chapter 2.3, 'Response messages'.

2.2.9 Syntax diagrams

Beside the syntax rules from the previous chapters you will find in the following a complete syntax diagram. Here you can see in detail which characters and separators are allowed at which position. The 'Program Data' are excluded to Figure 9. The numbers in parentheses reference the corresponding chapters.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

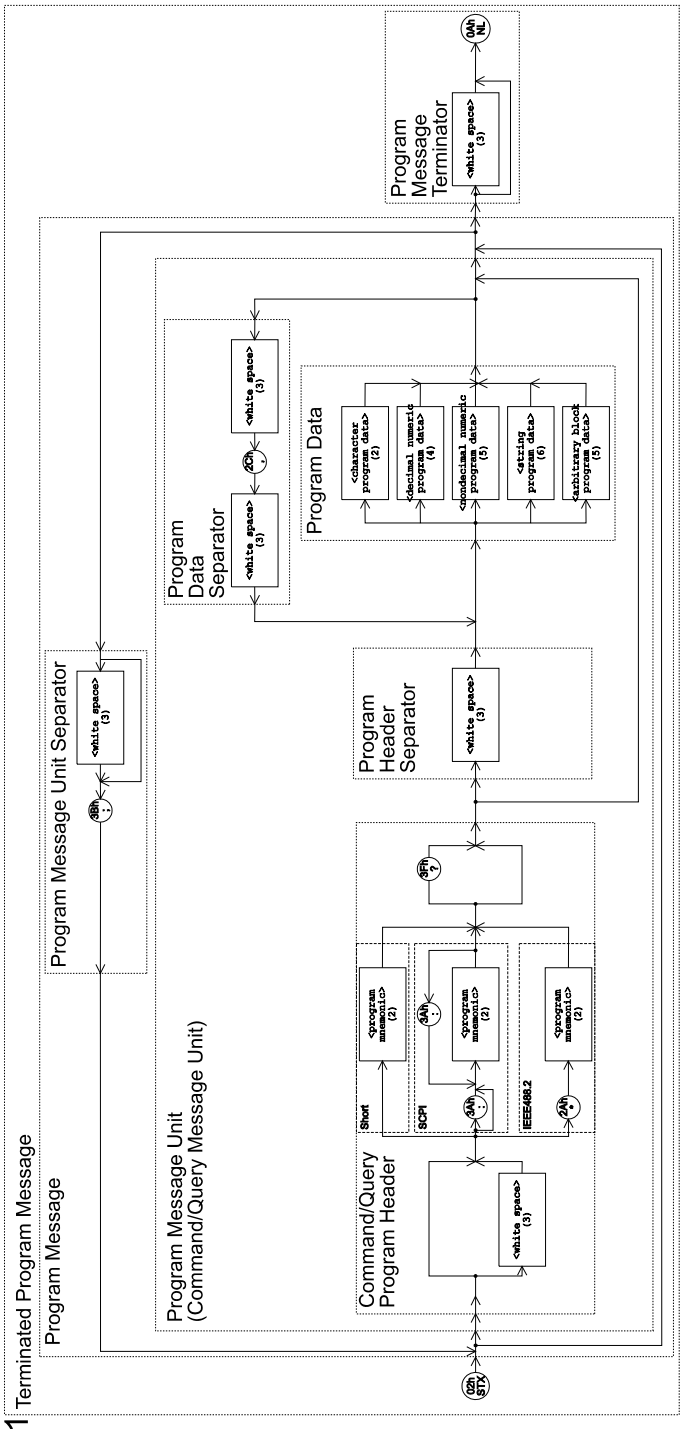

Figure 8: Overview Terminated Program Messages

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

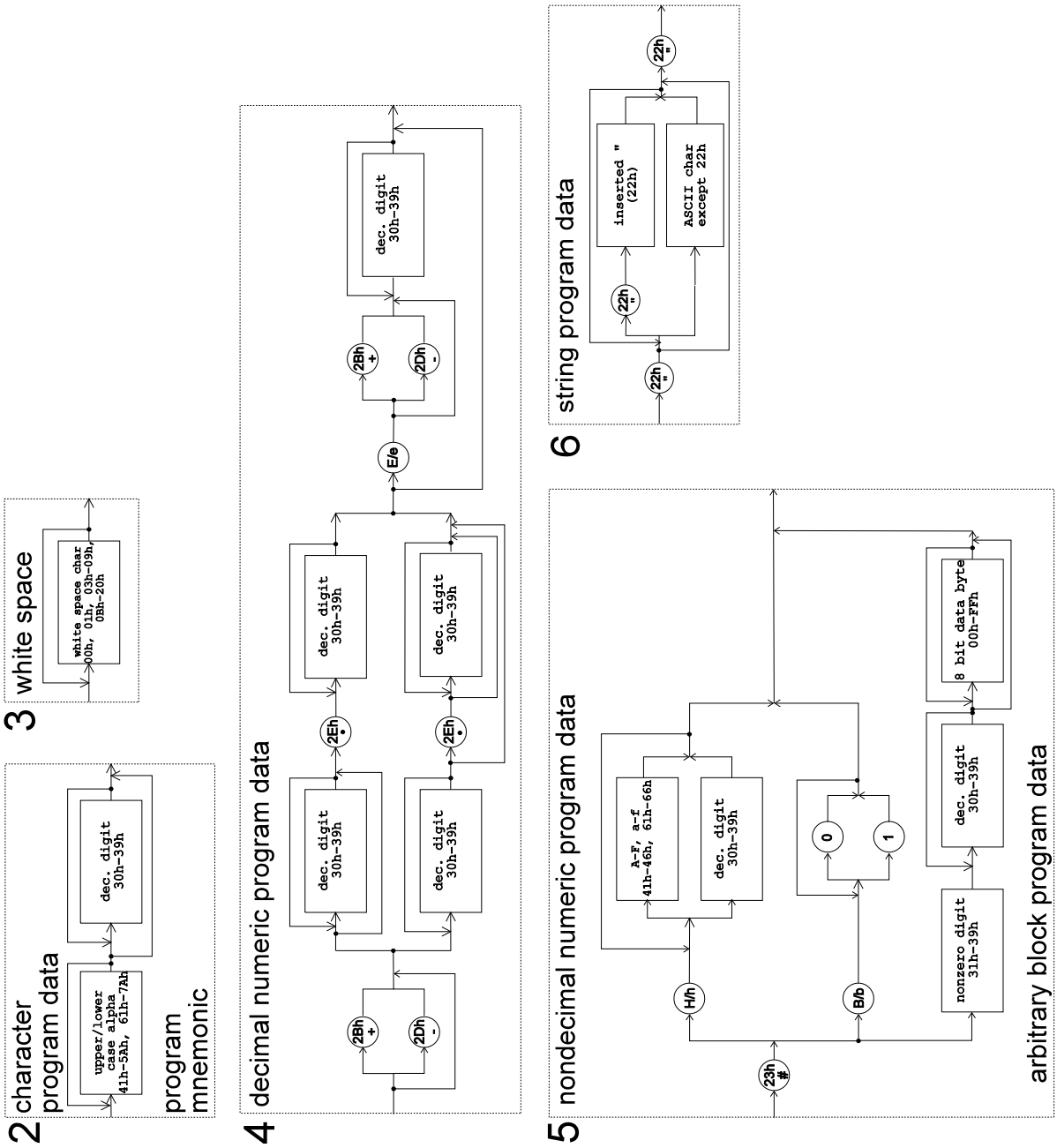

Figure 9: Overview Program Data

For example you can see in these diagrams, that a white space before a command is allowed, but not between `:` or `*` and the command itself (see section 1, 'Command/Query Program Header').

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

2.3 Response messages

A response message of the instrument can be composed of several values. It is terminated by an EOS (=end of string) symbol.

By default, numbers are returned as NR1 (integer, not to confuse with <NRi> parameters) or NR3 (float, not to confuse with <NRf> parameters) format (see 'Figure 10: Overview Terminated Response Message' for the exact syntax). Hexadecimal numeric response data and binary response data are not delivered by the ZES LMGs. But there are two other important data types which required some attention when programming.

In 'definite length arbitrary block response data' the EOS character can be included as valid data byte. It has to be ignored in this case, because it is part of the data. A detailed description follows in 3.2, 'Binary answers'.

String data are in principle also ASCII data, but they can contain EOS characters between the quotation marks. In this case the EOS code is also part of the data and not an EOS. String data are always sent in quotation marks ' " ', see Figure 10 section 13.
A practical example for sending and receiving a string is the formula editor of the LMG instruments.

Example: Formula editor
Instead of entering a formula directly at the instrument it is much more comfortable to send it by an interface command:

SCPI: :CALCulate:FORMula[:DEFine] <string>
SHORT: FORM <string>

The formula is passed inside a string (see chapter 2.2.8.3, '<string>').

SCPI: :CALCulate:FORMula "a=1;b=2;c=3;"
SHORT: FORM "a=1;b=2;c=3;"

By this string the variables a, b and c get the values 1, 2 and 3. This would be displayed in the formula editor of the instrument as

a=1;b=2;c=3;

As explained above, a `linefeed` (lf) is valid inside a string, even if EOS is also set to linefeed. If you used this character, the formula editor in LMG would then display:

a=1;
b=2;
c=3;

which is much easier to read. This formula string can also be read back. This is done in the following small Python example.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

```
#

#STRING.PY
#

# load ZES RS232 functions
import zes

#Open the serial interface:
LMG=zes.Open("COM1", "BAUD=38400 EOS=LF PROTO=RTS/CTS")

# Write formula without linefeed to instrument
zes.Write(LMG, "CALC:FORM \"a=1;b=2;c=3;\"")
# Request formula from instrument
zes.Write(LMG, "CALC:FORM?")
# Read answer from instrument
answer=zes.Read(LMG)
# Place answer on screen
print "Answer without linefeed and standard Read function is:\n"+answer

# Write formula with linefeed to instrument
zes.Write(LMG, "CALC:FORM \"a=1;\nb=2;\nc=3;\"")

# Request formula from instrument
zes.Write(LMG, "CALC:FORM?")
# Read answer from instrument with standard Read function
answer=zes.Read(LMG)
# Place answer on screen
print "\nAnswers with linefeed and standard Read functions are:\n"+answer
# This answer has terminated at first lf, so read the other ones:
# Read answer from instrument with standard Read function
answer=zes.Read(LMG)
# Output answer on screen
print answer
# Read answer from instrument with standard Read function
answer=zes.Read(LMG)
# Place answer on screen
print answer

# Request formula from instrument
zes.Write(LMG, "CALC:FORM?")
# Read answer from instrument with ReadQuota function for strings
answer=zes.ReadQuota(LMG)
# Place answer on screen
print "\nAnswer with linefeed and ReadQuota function for strings is:\n"+answer
# Write command to instrument
zes.Write(LMG, "GTL")

# Close connection
zes.Close(LMG)
print "\nProgramm finished"
```

Example 5: Program STRING.PY

How to start such a program and the general functions were already explained in 'Example 1: Program HELLO.PY'. Here we explain just the new things:

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

zes.Write(LMG, "CALC:FORM \"a=1;b=2;c=3;\"")	Send a string to the formula editor. Please note here that you have to send the quotation marks as part of a string. In Python (as well as in C/C++) you have to enter them as '\"' to prevent that the quotation mark is recognised as end of text of the zes.Write function.
zes.Write(LMG, "CALC:FORM?") answer=zes.Read(LMG)	Request the actual formula from the instrument. Read the answer from the instrument and store it in the variable 'answer'.
print answer zes.Write(LMG, "CALC:FORM \"a=1;\nb=2;\nc=3;\"")	Output this variable on the screen. Here additional linefeeds are sent to the instrument. They are entered as '\n' in the source code.
answer = zes.ReadQuota(LMG) print answer	Read the answer from the instrument and store it in the variable 'answer'. Due to the linefeeds in the stored formula the standard zes.Read function would terminate when the first linefeed occurs. For this reason the function ReadQuota was used. It was programmed according to the hints in 2.8.4.2, 'String data'.

Syntax diagram

Beside the response message rules from the above chapters you find following a complete syntax diagram. Here you can see in detail which characters and separators are allowed at which position in an answer from the instrument.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

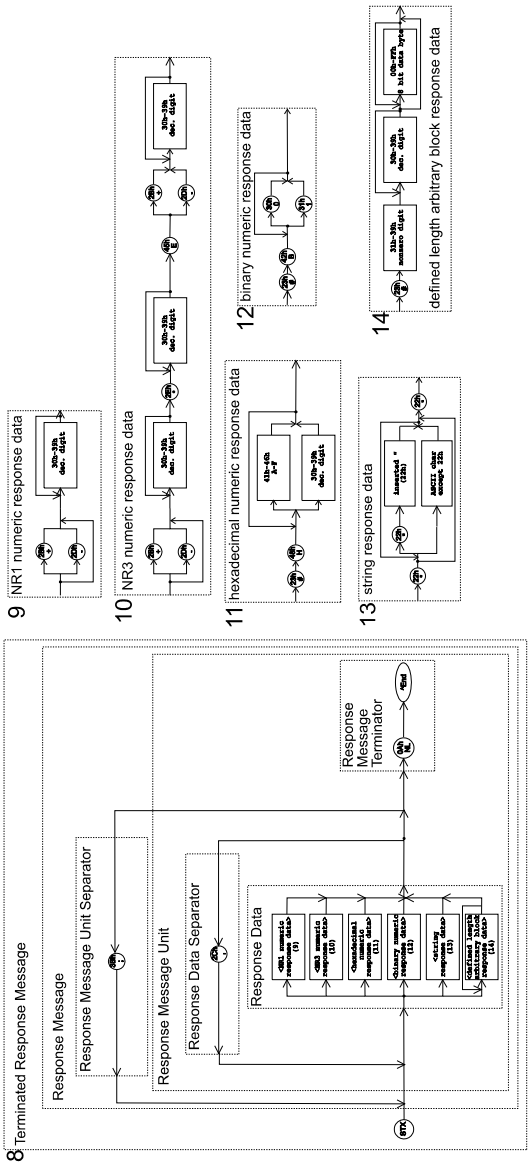

Figure 10: Overview Terminated Response Message

2.3.1 Invalid data, NaN

NaN, **N**ot **a** **N**umber, indicates a not defined number (e.g. not existing limit, frequency out of range ...). When transferring binary data (see 3.2, 'Binary answers') there is a special code for this value (conforming to IEEE754). Transferring in ASCII format NaN is represented by 9.91E37 (conforming to SCPI Standard).

The LMG instruments display NaN values as '--------'.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

2.4 Data flow, system structure

After the explanation of the syntax of the commands and prior to the basic interface functions this chapter should give you an overview of the complete communication system. Here you can see, which functions in the PC/instrument are involved and which way the data flows.

The transfer structure of the data is similar to the OSI model of usual computer networks:

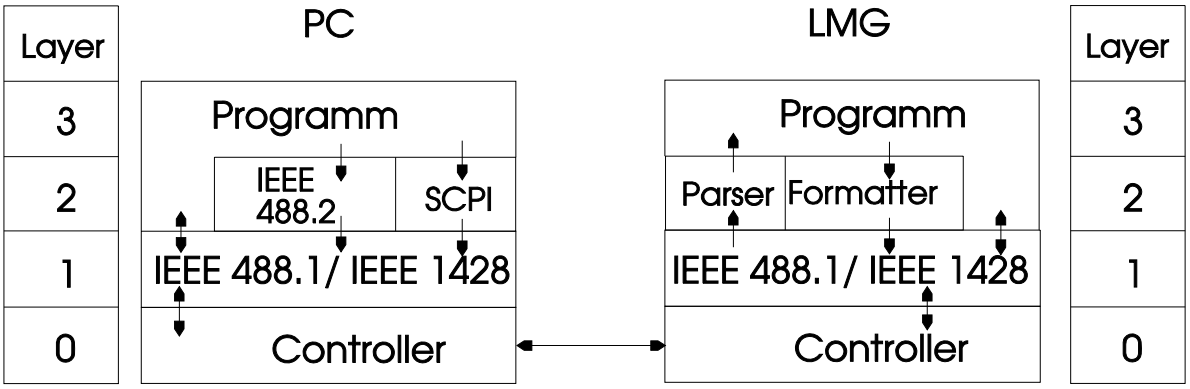

Figure 11: Transfer structure

Layer 0
This lowest layer is the 'physical layer'. It is the controller with the cables. This controller is directly affected by universal commands (IEEE488.1) or special RS232 Commands (defined in IEEE1428).

Layer 1
With the IEEE488.1 functions/commands the communication on the bus is controlled, e.g. which instrument(s) is/are addressed. This communication is done automatically between the controllers.

With addressed universal commands you communicate with one device (e.g. the command 'SDC' (Selected Device Clear) forces this single device to reset its interface). With non-addressed universal commands you communicate with all devices (e.g. the command `DCL` (Device Clear) forces all instruments to reset their interface).

IEEE1428 tries to simulate some of the IEEE488.1 commands. This is not completely possible, because there are too few control signals. But an important example is the `Break` of RS232 which is equal to the `SDC` command of IEEE488.1.

Layer 2
In this layer you find the IEEE488.2, SCPI and SHORT commands.
The IEEE488.2 common commands are always addressed commands. They start with a `*`, followed by three letters and sometimes a '?' question mark. In case of the '?' an answer from the instrument is expected (for details please see chapter 2.2, 'Program messages'). By

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

this all typical functions of an instrument are defined like `*IDN?` for identifying a device or `*RST` for resetting.

The SCPI and SHORT commands are device specific commands. They are used for setting up the instrument, requesting measuring values, ...

Layer 3
This is the software application itself (e.g. a terminal program or the program you are writing). This program is connected to layer 2 and layer 1. So it is the control centre for all functions. The program can execute all three types of commands:

- universal commands
- common commands
- device specific commands

The IEEE488 bus (as a point to point connection) and the RS232 have a nearly identical system overview. It is shown in Figure 12. The FIFOs 1-3 and 6-8 can be implemented optionally. Further on their size can vary greatly.

FIFOs 4 and 5 as well as the queues are implemented in the LMG instruments. Some instruments from other manufacturers have them implemented only partially or not at all.

The following example shows the typical data path from command to response, see also Figure 12.

Reading the AC voltage from the instrument
After the SHORT commands
`INIM;UAC?`
(INIM is the SHORT command for :INITiate:IMMediate. Further details follow in chapter 2.8.1.1, ':INITiate:IMMediate and INIM') have run through the FIFOs 1-3 they are transferred over the cable and stored in the input queue. The parser analyses the commands for correct syntax, parameters and so on. If the commands are valid, internal commands are generated and stored in FIFO 4. The execution unit forces the measuring unit to wait until the end of the currently running cycle (by the INIM command) and then to copy the measuring values into the interface buffer. The `Device Function` then takes the value of the AC voltage out of the interface buffer and stores it in FIFO 5. The response formatter converts this number into a format which can be read by the PC (usually ASCII). This data are now transferred to the output queue. Then the information is transferred via the cable and the FIFOs 6-8 into the program for further evaluation.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

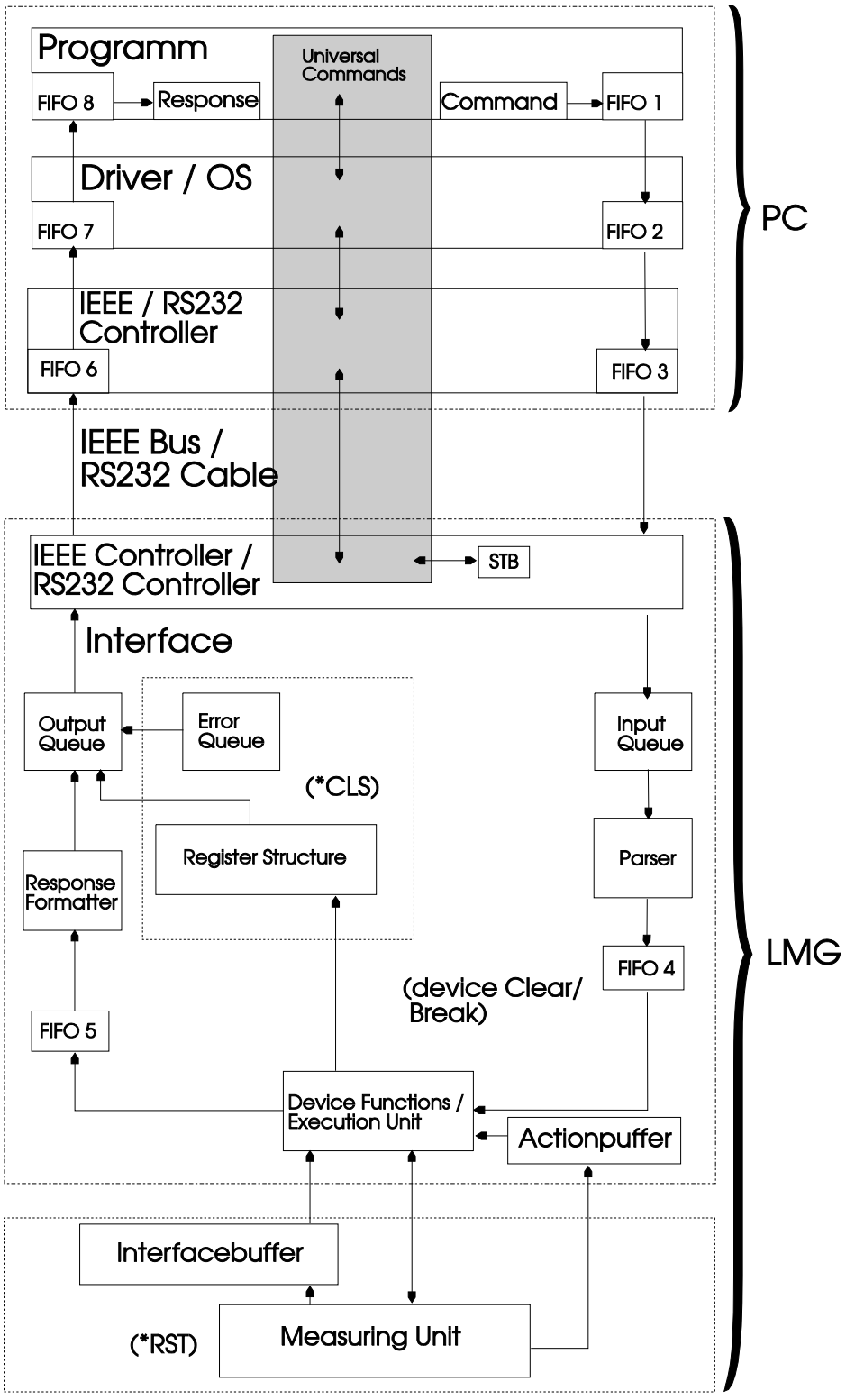

Figure 12: System overview of data flow

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

In Figure 12 the universal commands are grouped in a grey box. As described in chapter 2.1 they are not usual ASCII commands but are transferred via special states of the wires directly into the interface controllers. So they bypass all FIFOs! Keep this in mind, it is very important later on.

If for example the data flow is blocked by a wrong command it would be impossible to reset the instrument with any other command, because it would never reach the execution unit. But the universal commands are directly sent to the controller and can force the instrument to reset its interface.

The different units in the diagram above have their reset command in parenthesis. Some are universal commands (e.g. device clear), some are common commands (e.g. *RST). How to handle these commands and which sequence is important is explained in chapter 2.5, 'Opening the interface'.

2.5 Opening the interface

Before you have contact to the instrument, you have to decide, which interface you want to use: IEEE488 or RS232. This information is necessary in the LMG as well as in the PC. Both have to open the correct interface with corresponding parameters.

For opening you need some other information like baud rate, EOS for RS232 or instrument address for IEEE488. These parameters have to be set-up in the same way in PC and LMG.

Before starting the communication, it is strongly recommended to reset and initialise all data paths and the instrument itself (see following chapters).

The reset forces the affected unit to go into a defined state and to clear the buffers, queues and memories.

To reset the complete data path from the PC to the LMG (and back!) you have to reset different units in a defined sequence. This is described below.

Hint
Use the RTS/CTS protocol with RS232 interface. Then you have maximum safety for your data transfer. Most modern PCs (Windows as well as Linux based) are not real time systems. That means they can be blocked for such a long time, that the hardware FIFO input buffer can overflow. RTS/CTS protocol helps to minimise this problem.

2.5.1 Deleting the PC's output buffer

First you have to clear the output buffers of the PC for the case that they already contain any data. In detail these are the FIFOs 1-3. They are part of the used software, operating system, drivers and the IEEE488/RS232 controller itself.

They have to be deleted in the sequence of the normal data flow. This is very important to guarantee that there are really no old data left in the system. If you for example delete FIFO3 before FIFO2 some data could reach FIFO3 after deleting and that would cause

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

problems. This guarantees that an already reset unit does not become active until the programmer wants it.

To delete in the sequence of normal data flow is a general task when programming.

If you do not follow the correct sequence, the following can happen:

- Old data are interpreted as new data
- Commands seem to be misspelled and are rejected due to wrong characters in front of them
- Old commands are interpreted as new commands
- ...

2.5.2 Resetting the LMG interface

Next you have to reset the FIFO 4 and 5 as well as all units of the LMG interface with the `device clear` command (IEEE488) or `break` command (RS232).

These are universal commands. So they can bypass the usual data path and be received directly from the LMG. Internally the resetting and deleting of FIFOs is done in the usual way: Deleting input queue, resetting parser, ...

Additionally to the above action the following will happen:

- If you are in continuous mode (advanced topic, see 3.1.2, 'Continuous mode'), it will be stopped.
- If you are using the SHORT language, the language will change back to SCPI.

2.5.3 Deleting the PC's input buffer

One step which is often forgotten: Delete the input buffers of your PC!

In detail these are FIFOs 6 (IEEE488/RS232 Controller), 7 (OS or driver) and 8 (software). Take care to use the correct sequence here, too.

Now the complete data path is clear and ready for usual commands.

2.5.4 Resetting register structure and error queue

With the common command `*CLS` (Clear Status) the event register of all register structures is reset. Furthermore the error queue is cleared.

We recommend using this command even if it is not necessary. In case you use it, you can be sure, that all errors in the error queue which occur later on are caused by your program. This helps you to find the cause of the errors.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

The register structure is an advanced topic and explained in 3.3, ‘Control structures’

2.5.5 Resetting the measuring unit

Until now we have just reset and initialised the interface part of the instrument. The measuring unit itself is unchanged. So all measuring ranges, scalings, ... are unchanged until now.

As a last step we recommend to reset the measuring unit itself. This is done by the common command `*RST` (reset). This command is applied only to the measuring unit and does not influence any part of the interface.

What is the advantage of resetting the measuring unit? Let us assume that you just want to measure the voltage in the 6V range. So you initialise the instrument (without sending ‘*RST’), change to the 6V range and request the voltage. This can work, but not necessarily. If someone has manually changed the scaling you will get a wrong value. So you also would have to set the scaling, too. Then you should set the filters, too, and in principle all other parameters of the instrument just to be sure to know their setting. It is unknown how the LMG was set up previously.

With ‘*RST’ you are in a defined setting. Now you only have to change the values which are different from this defined setting. This is very simple, you do not have to send many commands and you can be sure not to have forgotten any parameter.

As told above this is just a recommendation.

2.5.6 Python Example

In the above Python programs we only used the ‘zes.Open’ function to open and initialise the interface. This was sufficient for the simple examples, but in general an initialisation should look like this:

```
#
# OPEN.PY
#

# load ZES RS232 functions
import zes

# load funtion Write_command_with_opc
import writeopc

def init(port, parameter):
   # Open the interface, clear output buffers
   handle=zes.Open(port, parameter)

   # Initialize the interface of the instrument
   # send ‘break’ if RS232 interface
   zes.Reset(handle)

   # Delete the input buffers of the PC
   zes.EraseBuffer(handle)
```


ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

```
# Reset register structure and error queue
writeopc.Write_command_with_OPC(handle, "*CLS")

# *RST could take some time for execution, so the timeout
# is increased
zes.SetTimeout(handle, 1000)
# Reset the measuring unit
writeopc.Write_command_with_OPC(handle, "*RST")
# Reduce again to standard value
zes.SetTimeout(handle, 500)

return handle
```

Example 6: Program OPEN.PY

Here we wrote a function which takes as parameters the port and its parameters. The handle to the opened communication port is returned. This function cannot be executed standalone. It will be used later on (e.g. in 'Example 10: Program OPC.PY').

2.6 Closing the interface

Before you close the connection to the instrument you should

- Leave the continuous mode (advanced topic, see 3.1.2, 'Continuous mode'), if you use it.
- Read out all unread data from the instrument or reset the interface (see 2.5.2, 'Resetting the LMG interface').
- Clear the PC's input buffer (see 2.5.3, 'Deleting the PC's input buffer').
- Request all error messages with the :SYSTem:ERRor:ALL? (SCPI) or ERRALL? (SHORT) command. If you have cleared the error queue (see chapter 2.5.4, 'Resetting register structure and error queue`) all displayed error messages were caused by your program. Do this even if your program seems to work correctly! Try to find the cause of any shown error message.
- Set the instrument back to 'local' state so a user can work with it. To do this you can use the IEEE488.1 universal command 'go to local' or the RS232 command 'GTL'. Take care that this is the last command, else the instrument will change back to remote mode.

2.6.1 Python Example

In the above Python programs we used just the 'zes.Close' function to close the interface. This was sufficient for the simple examples, but in general a closing should look like this:

#

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

```
# CLOSE.PY
#

# load ZES RS232 functions
import zes

def close(handle):
   # Initialize the interface of the instrument.
   # By this the continuous mode (if used) is stopped
   # and all buffers in the LMG are cleared.
   # Further on the language is changed back to SCPI
   zes.Reset(handle)

   # Delete the input buffers of the PC
   zes.EraseBuffer(handle)

   # Request the error messages.
   zes.Write(handle, "SYST:ERR:ALL?")

   # Read in the error messages
   error = zes.Read(handle)
   # check if any error occured
   if error != "0,\"No error\"":
      # output them on the screen
      print "Instrument sent following error message(s):\n" + error

   # Change back to local mode
   zes.Write(handle, "GTL")

   # Close the device
   zes.Close(handle)
```

Example 7: Program CLOSE.PY

Here we wrote a function which takes the handle as parameter. This function cannot be executed standalone. It will be used later on (e.g. in 'Example 10: Program OPC.PY').

2.7 Writing to the interface

If the interface is open and you send the first byte to the LMG, it changes to the `remote` state. Now you cannot change any parameters at the LMG itself but only via the interface. This is to prevent double entries at the same time and to protect the software from unexpected changes in the instrument set-up.

The write function to the instrument can be realised in two very common ways:

- Most write functions work with the so called ASCII-Z format (Z=Zero). These expect the start address of the data to be sent as input. This routine sends out data until the character with the value 0h (not '0'=30h!) is reached. Then the function returns. This

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

kind of function is easy to handle, because you do not need to care about the length. The disadvantage is that you cannot send a zero character.

- Some write functions require the starting address and length of the data to be sent. This routine then sends exactly this data without any interpretation. This type of functions is more complex to handle because the programmer has to determine the length of the data. The advantage is that you can send every byte to the instrument that you want, including a 0h.

For convenience and safety each method should automatically add the EOS character at the end.

Hints

1. The LMG instruments use only ASCII data as input. So no zero character is necessary and you can use a write function of the first type with ASCII-Z format.
2. In most cases it is a good solution, if the write function adds the EOS (and EOI) automatically (see 2.7.2, 'EOS and EOI'). If your write function does not do this automatically, write your own!
3. Do not use the EOS character to detect the end of a message (see 2.7.2, 'EOS and EOI')!

2.7.1 Data format

The LMG instruments use only ASCII data as input (see 2.2, 'Program messages'). Only the data output can include binary data, see 3.2, 'Binary answers'.

2.7.2 EOS and EOI

To terminate a data transfer you have to send the EOS (end of string) character.

IEEE488

Additionally to the EOS character IEEE488 transfers the EOI (end of identification) signal. The EOS is always set to <lf>.

RS232

RS232 can use several types of EOS:

- <lf> (linefeed, 0Ah)
- <cr> (carriage return, 0Dh)
- <cr><lf> (0Dh 0Ah)

You have to decide which type you want to use.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

Caution
The EOS code can be the end of a message, but does not have to be. Let us assume you want to transfer a string to the instrument (see 2.2.8.3, '<string>'). Inside the quotation marks you are allowed to send every character, even the EOS! So when using the IEEE488 interface make sure that your write function does not set EOI when it detects any possible EOS character, but only when it detects an EOS which has the meaning of EOS!
Of course the LMG instruments also detect only real EOS characters as EOS but no EOS characters inside a string! So you can send every character inside a string.

Hint
1. When using RS232 use <lf> as EOS, too. The other possibilities exist only for compatibility with very old computer systems or terminals.
2. Use a write function which adds the EOS (and EOI at IEEE488) automatically. So it cannot be forgotten.

2.7.3 Timeout

RS232 as well as IEEE488 interfaces use a so called timeout (RS232 only if RTS/CTS protocol is used). It is used to detect if the write function blocks too long and terminates it with an error message. For example you want to send some data to the instrument but it cannot receive them for any reason (cable is not plugged in, ...). In this case a write function without timeout would wait forever and the PC program would hang.

To solve this problem a write function with timeout should be used. If the instrument does not receive the data in a usual time, an error message is generated and the function is left. Then you can try to initialise the interface of the instrument again (see 2.5.2, 'Resetting the LMG interface'. Due to the fact that universal commands are used here a pure transfer problem can be solved (grey path in 'Figure 12'). These functions should use a timeout, too. If this also fails, you have to check the hardware connections and/or the settings in your program and of the LMG.

Example
Assume you have a cycle time of 10s. Then due to a program failure you send a `INIM;UTRMS?` (SHORT) command to the LMG every 100ms. So after a 100 requests the first one is answered, after another 100 the second one and so on. The FIFO 4-1 will overflow after a while and the data transfer to the instrument is blocked. With a timeout, you can reset the instrument and search for the cause. Without a timeout your program is blocked.

Recommendation
1. Use a timeout when sending data. If it fails there is a problem with the program or the instrument.
2. To minimise programming effort, you can write your own write function which supervises the timeout and generates a correct error message in case that it happens.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

2.7.4 Usage of '*OPC?'

The commands are executed in the sequence you send them. If you first change a range (long execution time) and then request a value (short execution time) you can be sure, that the request of the value is not executed before the range has been changed.

If you try to change all voltage and current ranges in one message and then immediately request a value, this can take much more time than expected (one cycle, see chapter 2.8.1.1, ':INITiate:IMMediate and INIM'). By this it can happen that you run into a timeout when reading the request.

A solution could be the `*OPC?` common command. This command returns a simple '1' when it is executed. So if you split several commands with longer execution time into several messages, add a ';*OPC?' to each message and wait after each message until the '1' is returned you should have no problems. The following example (SHORT commands, for LMG450) changes the mode of the ranges to 'manual':

```
IAM1 0;IAM2 0;IAM3 0;*OPC?
```

By this you wait for the execution of the few commands and the execution times of all commands are not added. You can work with a usual short timeout.

Hint
You can write your own `Write_with_OPC` function which automatically adds the ';*OPC?' and waits until the '1' is returned:

```
#
# WRITEOPC.PY
#


#load ZES RS232 functions
import zes

# Function Write_command_with_OPC
# This function sends a command to the instrument, adds an
# ';*OPC?' to this command and waits, until the instrument answers
# with a '1'.

def Write_command_with_OPC(instrument, text):
   zes.Write(instrument, text+";*OPC?")
   answer = ""
   while answer != "1":
      answer = zes.Read(instrument)

# End of function
```

Example 8: Program WRITEOPC.PY

In this example the bare function is realised without any timeout supervision.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

This small function, together with the above developed small ones (see 'Example 6: Program OPEN.PY' and 'Example 7: Program CLOSE.PY') are combined in a tools collection:

```
#
# G_TOOL.PY
#

execfile ("open.py")

execfile ("close.py")

execfile ("writeopc.py")
```

Example 9: Program G_TOOL.PY

This tools collection is used in the next example to synchronize the program exactly. In this case a beep is played by the LMG when the change of the command language has been successfully performed.

```
#
# OPC.PY
#

# load ZES RS232 functions
import zes

# load tool collection of programmer's guide
import g_tool

# Open the serial interface, clear output buffer,
# initialize data path and instrument.
LMG=g_tool.init("COM1", "BAUD=38400 EOS=LF PROTO=RTS/CTS")

# Change to SHORT language
g_tool.Write_command_with_OPC(LMG, "SYST:LANG SHORT")
# Let the instrument beep
g_tool.Write_command_with_OPC(LMG, "BEEP")

# Close the interface. It is assumed, that the handle of the LMG
# instrument is called 'LMG' and the RS232 interface is used.
g_tool.close(LMG)
print "Programm finished"
```

Example 10: Program OPC.PY

Note

The above Write_command_with_OPC function cannot be used when you request any values. It would wait until a '1' return which might never happen. This function should only be used when sending commands without answer.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

2.7.4.1 Command sequence

As told above, the usual common and device specific commands are executed in the order they are sent to the instrument. But universal commands can overtake them. Take a look at the next example which demonstrates the problem:

```
#
# TEST.PY
#

# load ZES RS232 functions
import zes

# load tool collection of programmer's guide
import g_tool

# Open the serial interface, clear output buffer,
# initialize data path and instrument.
LMG=g_tool.init("COM1", "BAUD=38400 EOS=LF PROTO=RTS/CTS")

zes.Write(LMG, "hello")

# Close the interface. It is assumed, that the handle of the LMG
# instrument is called 'LMG' and the RS232 interface is used.
g_tool.close(LMG)
print "\nProgramm finished"
```

Example 11: Program SEQ1.PY

After opening the interface the (non-existing!) command 'hello?' is sent to the instrument. The close.shutdown function (see 'Example 7: Program CLOSE.PY') should not output an error message. But if your computer is not a very slow one you will get no error message. The reason is, that the first command in 'Example 7: Program CLOSE.PY' is a universal command. This is executed so fast after the wrong command 'hello' has been sent, that the input buffer of the instrument is deleted before the command was parsed and the error detected (see also 'Figure 12: System overview of data flow').

There are two solutions:

- You wait for a certain time after sending 'hello' to be sure it was 'executed' which means in this case the error was detected. But delays in a program are always a bad solution, because the required duration of the timeouts could change and this would cause trouble.
- The other way is to get a notification from the LMG, when a command has been finished: 2.7.4, 'Usage of '*OPC?''

The following example uses the last method:

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

```
#

# SEQ2.PY
#

# load ZES RS232 functions
import zes

# load tool collection of programmer's guide
import g_tool

# Open the serial interface, clear output buffer,
# initialize data path and instrument.
LMG=g_tool.init("COM1", "BAUD=38400 EOS=LF PROTO=RTS/CTS")

g_tool.Write_command_with_OPC(LMG, "hello")

# Close the interface. It is assumed, that the handle of the LMG
# instrument is called 'LMG' and the RS232 interface is used.
g_tool.close(LMG)
print "Programm finished"
```

Example 12: Program SEQ2.PY

Here the program execution continues at once after the '1' was received. By this you can be sure that the wrong command has been executed.

Of course similar problems could also happen with valid commands!

2.8 Reading from the interface

You can only read from the interface, if you have requested values from the instrument. The instrument will usually not deliver data on itself (for the single exception see 3.1.2, 'Continuous mode') but only on a single request.
The time from your request to the delivery of the data can vary a lot, depending on your request method (see 2.8.1, 'Single request').

Usually it is not a good idea to enter the read function directly after the request. The reasons are:

- Usually you have a timeout of 0.5s (see also 2.8.3, 'Timeout'). If an answer takes 3s (see example in 2.8.1.1, ':INITiate:IMMediate and INIM'), the read function will return with a timeout error. To solve the problem you have several possibilities: Extend the timeout or check if the instrument wants to send data (see below)
- Depending on how the read function is implemented on your computer, the system is blocked, while you are in this function. A graphical interface is not useable during this

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

time! To prevent this problem, use a non-blocking 'read' function or check if the instrument wants to send data (see below).

Checking if instrument could send data
To check if you can enter the read function you should

- check (only possible if using IEEE488) the MAV (message available) bit by serial polling (universal command, can be executed anytime) before entering the read function (see 3.3.1, 'Status Byte Register') or
- check (only possible if using RS232) if there are any bytes coming from the instrument into the PC's input buffer before entering the read function

2.8.1 Single request

When using a single request, the measuring data are only copied to the interface buffer (see Figure 12) when this is forced by the programmer. When a copy is forced, all measuring values are copied. This has some advantages and consequences:

- All values were measured in the same cycle, so they physically belong together.
- You control when the interface buffer is updated. So you have as much time as you want to read out the values. They remain there unchanged.
- If you forget to copy the values, you will read old values! Avoid this.

2.8.1.1 :INITiate:IMMediate and INIM

'INIM' is the SHORT command for the SCPI command `:INITiate:IMMediate`. This command forces the instrument to wait until the end of the actual cycle and then copy all measuring data into the interface buffer (see Figure 12).

The values are copied at the end of a cycle. In the worst case (when you send the command at the begin of a cycle) it can take up to a complete cycle time until the values are copied (at 3s cycle time it can take 3s ±some milliseconds!). If you sent the command and then immediately entered the 'read' function to get data from the interface this function would also take up to 3s. This can cause several problems. Some strategies for solving these problems where shown above.

Please send only one INIM per message to the instrument. Send a second one not before the first one is finished.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

2.8.1.2 :INITiate:COPY and COPY

If you need the last measuring values before the actual cycle, you can force the instrument to copy the values immediately with the 'COPY' (SHORT) or `:INITiate:COPY` (SCPI) command.

2.8.2 :FETCh, :READ and SHORT commands

The two branches of the SCPI tree are used to read out measuring values. The difference is that ':FETCh' will read data from the current interface buffer (without copying). ':READ' waits until the end of the currently running cycle, copies new data to the interface buffer and reads then data from the new buffer. So ':READ' is a combination of the ':INITiate:IMMediate' (see 2.8.1.1) and the ':FETCh' command.
If you request the same value twice with two :FETCh commands you get the same values of the same cycle, because the interface buffer did not change. For example:

FETC:DC?;:FETC:DC? would not make any sense, because you would get the same value twice.

If you request the same value twice with two :READ commands (e.g. :READ:DC?;:READ:DC?) you get two different values of two different cycles. This can cause problems for example with following request:

:READ:VOLTAGE:DC?;:READ:CURRENT:DC?
The two values you get for Udc and Idc are measured in different cycles!

A usual request looks like this:
:READ:VOLTAGE:DC?;:FETC:CURRENT:DC?
In this case the instrument finishes the current cycle, copies the values for the interface and returns the two requested values. These two values are measured in the same cycle!

SHORT Commands
The SHORT commands perform equal to the ':FETCh' commands (which means there is no 'INIM' (see 2.8.1.1) performed!). So if you want to perform the last example with SHORT commands you have to enter
INIM;UDC?;IDC?

Everything explained above is demonstrated in the following example. Please connect a voltage and current to the channel 1 of your instrument and start the program.

```
#
# GET_DATA.PY
#

# load ZES RS232 functions
import zes

# load tool collection of programmer's guide
import g_tool

# Open the serial interface, clear output buffer,
# initialize data path and instrument.
```


ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

```
LMG=g_tool.init("COM1", "BAUD=38400 EOS=LF PROTO=RTS/CTS")

# Extend timeout to 2000ms
zes.SetTimeout(LMG,2000)
# Skip first measuring cycle after reset, it could contain
# invalid values
g_tool.Write_command_with_OPC(LMG, ":INIT:IMM")

print "\nIdentical values from same cycle"
print ":FETC:TRMS?;:FETC:TRMS?                -> ",
zes.Write(LMG, ":FETC:TRMS?;:FETC:TRMS?")
print zes.Read(LMG)

print "\nDifferent values from different cycles"
print ":READ:TRMS?;:READ:TRMS?                -> ",
zes.Write(LMG, ":READ:TRMS?;:READ:TRMS?")
print zes.Read(LMG)

print "\nValues from different cycles"
print ":READ:VOLTAGE:TRMS?;:READ:CURRENT:TRMS? -> ",
zes.Write(LMG, ":READ:VOLTAGE:TRMS?;:READ:CURRENT:TRMS?")
print zes.Read(LMG)

print "\nValues from same cycles"
print ":READ:VOLTAGE:TRMS?;:FETC:CURRENT:TRMS? -> ",
zes.Write(LMG, ":READ:VOLTAGE:TRMS?;:FETC:CURRENT:TRMS?")
print zes.Read(LMG)

g_tool.Write_command_with_OPC(LMG, ":SYST:LANG SHORT")
print "\nValues from same cycles, SHORT language"
print "INIM;UTRMS?;ITRMS?                    -> ",
zes.Write(LMG, "INIM;UTRMS?;ITRMS?")
print zes.Read(LMG)


# Close the interface. It is assumed, that the handle of the LMG
# instrument is called 'LMG' and the RS232 interface is used.
g_tool.close(LMG)
print "\nProgramm finished"
```

Example 13: Program GET_DATA.PY

2.8.3 Timeout

There are two possible places for a timeout control:

- While reading the data itself. This timeout can be programmed with a fixed time, when it is used after checking the MAV bit. A usual value here is 0.5s.
- While checking the MAV bit (IEEE488) or if data are in the input buffer (RS232).
  The timeout for this check should depend on the cycle time, for example 1.5 times longer than the cycle time.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

Example
You send a message to the instrument, that you want to read a value. The cycle time is assumed to be 5s, the timeout for the read function is 0.5s. In this case the timeout for checking the MAV bit should be set to at least 7.5s.

2.8.4 EOS and EOI

At the end of a data transfer the LMG adds the EOS (end of string) to the message. When using IEEE488 the EOI signal is set, too.

When you get pure ASCII data from the LMG it might be sufficient to check for EOS to terminate a received message. But there are also other output formats which can contain the EOS as valid character (see chapter 2.2.8.3, '<string>' and 2.8.4.1, 'Binary data'). With IEEE488 you can check for EOI instead of EOS, but with RS232 you have to watch the received data very carefully.

If possible, try not to mix different data types in one request like in the following (SHORT) example

```
UTRMS?;FORM?
```

By this you request the TRMS value of voltage as number in ASCII format and the current formula as string. The answers are just separated by a semicolon and are difficult to interpret.

It is simpler to write two read functions, one for numbers and one for strings. Then you call the required one. The two separate requests should then look like:

```
UTRMS?
FORM?
```

2.8.4.1 Binary data

The format of the binary data is described in chapter 3.2, 'Binary answers'. If you receive binary data, your read function must not stop if receiving an EOS, but it must check if this EOS is inside a binary block (then it has to be ignored) or if it is outside (then it is a real EOS). With IEEE488 it is sufficient to wait for EOI. With RS232 you will have to interpret the received data. Then count how many binary data follow and for this number of bytes you have to deactivate EOS checking.

2.8.4.2 String data

If you receive string data you have to work in a similar way as with binary data. Your read function must not stop when it is receiving an EOS, but it must check if this EOS is inside the string (then it has to be ignored) or if it is outside (then it is a real EOS). With IEEE488 it is sufficient to wait for EOI. With RS232 you will have to interpret the received data. The string is between a starting and an ending quotation mark (see also paragraph 13 in Figure

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

10). A quotation mark inside a string is defined as a double quotation mark. While inside the string you have to deactivate EOS checking!

2.8.5 Buffer and real time

In general the LMG instruments are real-time instruments. They measure without any gaps and also have to send without any gaps. But in a typical environment this is impossible and the instruments therefore have implemented buffers to prevent data losses if a PC needs a microsecond. But a FIFO can only work, if the average data output is at least as fast as the average data input. If this is not true, the FIFO will overflow sooner or later depending on the size.

So please take care, that you request only as many data as the interface can handle and your PC can retrieve in a certain time. If you request more data the FIFO will overrun after a while.

Especially when using the continuous mode (see chapter 3.1.2, 'Continuous mode') it can happen with a slow computer (or a slow interface like RS232) that you get a buffer overrun. In a real-time system each part of the chain has to support real time!

3 Advanced programming

3.1 Automatic request

In opposite to the single request (see 2.8.1, 'Single request') you have the possibility to let the LMG

- copy all values automatically to the interface buffer after a cycle
- send a selection of them to the response formatter
- output them without an explicit request

This is done in the so called continuous mode. First you have to select with the `:TRIGger:ACTion` (SCPI) command, which values should be output. Then you enter the continuous mode with the ':INITiate:CONTinuous ON' (SCPI) command. Starting from now, all requested values are output automatically after the end of a measuring cycle. You do not have to send any messages to the instrument.

The advantages of this method are:

- You do not have to send the same requests to the instrument again and again. The data path can be used for the interesting measuring data. That saves a lot of transfer capabilities and time in the PC (for sending) as well as in the LMG (for parsing).
- You never miss any data by a request coming too late from the PC.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

- In combination with the binary data output (see 3.2) you get a highly efficient system.

3.1.1 :TRIGger:ACTion and ACTN

The ':TRIGger:ACTion' (SCPI) or 'ACTN' (SHORT) command selects the values which should be output when the continuous mode is active and a cycle has finished. All commands behind the semicolon ';' after the 'action' command until the end of the message are stored in the action buffer (see Figure 12). When a cycle is finished, the measuring values are copied to the interface buffer and the execution unit executes the commands which are stored in the action buffer.

Example

SCPI: :TRIGger:ACTion;:FETCh:VOLTage:TRMS?;:FETCh:CURRent:TRMS?

SHORT: ACTN;UTRMS?;ITRMS?

These commands define, that the TRMS values of voltage and current will be output after each cycle.

Note!

Do not use any 'INIM' (SHORT) or ':READ' (SCPI) commands after the action command. The data are copied automatically.

3.1.2 Continuous mode

The syntax to enter and leave the continuous mode is `INITiate:CONTinous <NRi>` (SCPI) or `CONT <NRi>` (SHORT).

Using `CONT 1` enters the mode, `CONT 0` leaves it. For better readability you can also write `CONT ON` and `CONT OFF`. The continuous mode is also stopped when sending a break (RS232) or interface clear (IEEE488).

Hint

Do not forget to leave this mode, when your program has finished. Otherwise the instrument will continue to output data. This could cause problems with initialization when starting a new program (see also 2.5.2, 'Resetting the LMG interface').

Recommendation

If you send `CONT OFF` at the end of a cycle it is possible, that the commands in the action buffer are executed before the CONT OFF is executed. So you will have data in the output queue. Make sure, that there are no data from FIFO 5 up to your program. If you forget to read out or clear all buffers it can happen that you get old data after your next request!

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

3.2 Binary answers

Binary data are defined in section 14, `definite length arbitrary block response data` in Figure 10. The output starts always with a `#`. The following single digit defines the number of digits which define the length of the following data. For example '#500012` means the following:

#	Binary message
5	The following 5 digits define the length of the following data
00012	After these 5 digits, which define the length of the binary data, 12 data bytes follow. If there is an EOS character inside these 12 bytes it has to be ignored!

The main advantages of binary data transfer are:

- Less bytes per value. Due to the header (#5000...) this advantage comes into effect if you transfer more than one value. This is much faster in transfer. For example a float number will be transferred in binary as 4 bytes. As ASCII (e.g. 1.23456e+7) it would be 10 bytes and more.
- The numbers do not have to be converted from internal binary format to ASCII format in the LMG and at the PC you do not have to do the opposite operation. This is much faster. Further on the accuracy of these numbers is a little bit better. The four byte binary numbers have a resolution of about 8-9 digits, the ASCII data just 5-6 digits.
- The data can be read directly into a 'struct' if you are programming for example in C/C++. This is very fast and efficient. You do not have to copy the values.
- All numerical values of the LMG instruments are transferred as long or float numbers, so they always have 4 bytes.
- In combination with the automatic request (see 3.1) you get a highly efficient system.

The next byte after a block of binary data is either an '#' to start a new block or an EOS.

Please note
A number of binary data can be separated into several blocks and transferred. In this case the next byte after a block is the start of a new block '#'. All this data together are the answer of one request. Several messages (each ending with EOS) are answered separately (each ending with EOS). Example:
#500008xxxxxxxx could also be send as #500005xxxxx#500003xxx.

Further hints for writing a read function for binary data you can find in 2.8.4.1, 'Binary data'.

To switch from standard ASCII answers to binary answers you have to use the commands

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

SCPI: :FORMat:DATA <NRi>

SHORT: FRMT <NRi>

So with FRMT PACKED you enable the binary output, with FRMT ASCII you switch back to standard output.

Note
When resetting the interface of the instrument (see 2.5.2, 'Resetting the LMG interface') the output format changes back to ASCII.
The following example shows, how simple it is to handle binary data in python:

```
#
# BINARY.PY
#

# load ZES RS232 functions
import zes

# load tool collection of programmer's guide
import g_tool

# load function to handle binary lists
import struct

# Open the serial interface, clear output buffer,
# initialize data path and instrument.
LMG=g_tool.init("COM1", "BAUD=38400 EOS=LF PROTO=RTS/CTS")

# Change language to SHORT
g_tool.Write_command_with_OPC(LMG, "SYST:LANG ZES")

# Skip first measuring cycle after reset, it could contain
# invalid values
g_tool.Write_command_with_OPC(LMG, "INIM")

zes.SetTimeout(LMG, 1000)

zes.Write(LMG, "INIM;UTRMS?;ITRMS?;P?")

# Read data as ASCII
answer = zes.Read(LMG)
print "The ASCII answer:"
print answer

# Change to binary data output
zes.Write(LMG, "FRMT PACKED")
# Request same values from SAME interface buffer
zes.Write(LMG, "UTRMS?;ITRMS?;P?")
# Read data as Binary
answer = zes.ReadBinary(LMG)
# Interpret them as a list of 3 float numbers
a = struct.unpack('fff', answer[0:12])
print "\nThe converted binary answers"
# output 1st element
print a[0]
```


ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

```
# output 2nd element
print a[1]
# output 3rd element
print a[2]

# Close the interface. It is assumed, that the handle of the LMG
# instrument is called 'LMG' and the RS232 interface is used.
g_tool.close(LMG)
print "\nProgramm finished"
```

Example 14: Program BINARY.PY

3.3 Control structures

In this chapter we describe the connections and functions of the control structure. It consists of a register structure and several queues. This structure is used for supervising the instruments. It represents errors and current states of the instrument. Figure 13 shows the complete structure.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

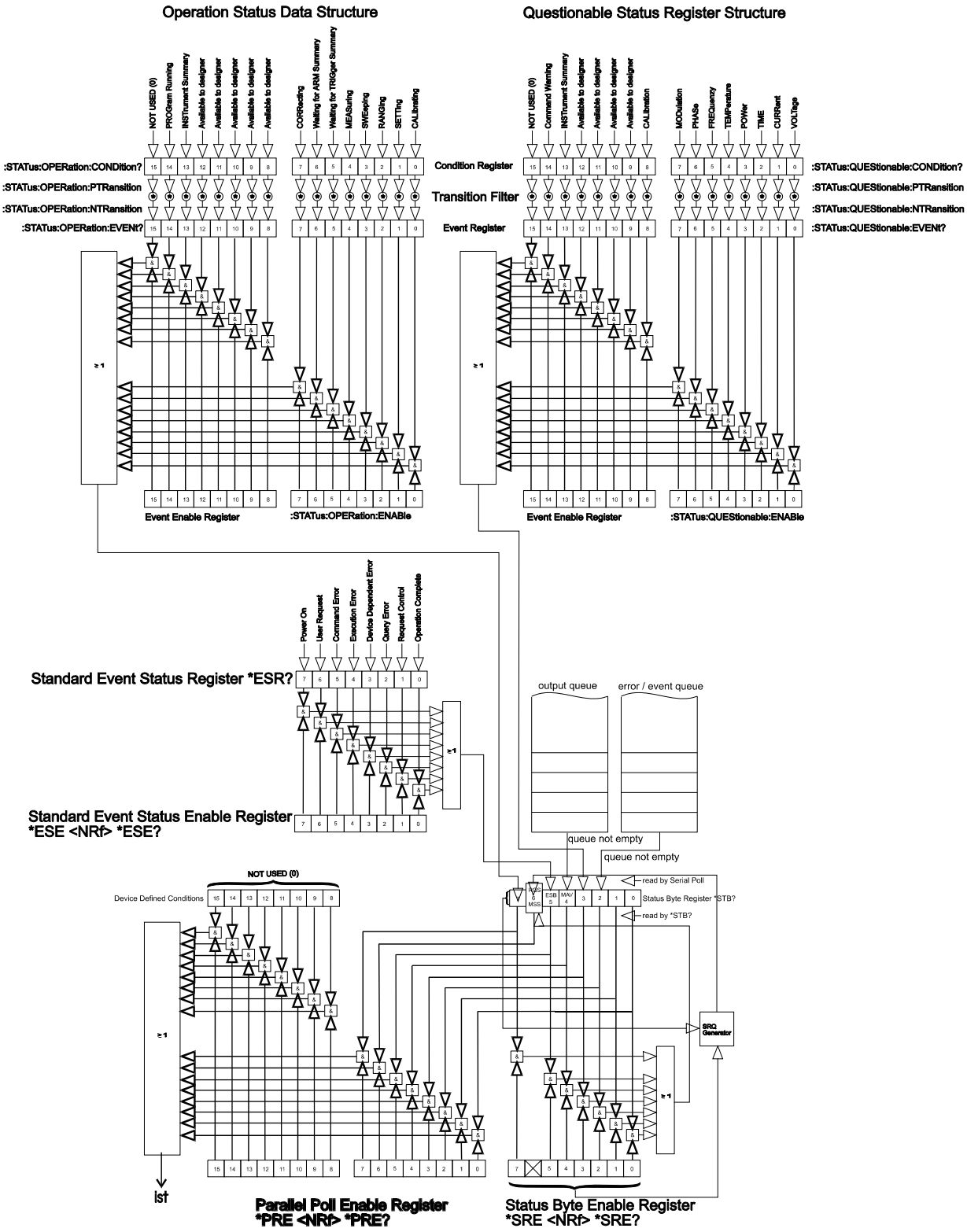

Figure 13: Control Structure Overview

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

The centre of the total control structure is the status byte register (STB) which collects the states of all other registers and queues. Using the **IEEE**488 interface you can request the content of the STB via universal commands (layer 1). So you can get its value at any desired time.

3.3.1 Status Byte Register

The status byte register collects different states of the instrument. It is not possible to write into this register.

One of the most important bits (when transferring via IEEE488) is the 'MAV' (bit 4). This bit is part of the handshake, if you use serial poll of IEEE488 (see chapter 2.8, 'Reading from the interface'). Instead of from layer 1 you can access it also from layer 3 with the common command `*STB?`. For RS232 communication this is the only way to get its content, but in this case the MAV bit will always be set.

The bits of the status byte register

Bit 0/1: These bits can be used for any purpose. They are unused in the ZES instruments (SCPI definition).

Bit 2: Query error bit (QYE). If this bit is set one or more error messages are available in the event / error queue (SCPI definition).

Bit 3: The set QUES bit indicates that the summary message of the questionable status register is set. So there is information about questionable measuring values inside it (SCPI definition). LMG95(e), LMG450 and LMG500 do not output questionable data, so this is never set.

Bit 4: If the MAV (message available) bit is set, you can get information from the output queue. This bit should be checked before you read from the IEEE488 bus to prevent timeouts (IEEE definition).

Bit 5: The set ESB (event status bit) indicates that the summary message of the standard event register is set. So there is information inside (IEEE definition).

Bit 6: This bit is different, depending on the way you request it:
If requested via the common command '*STB?', the MSS (master summary status) bit is returned. The MSS bit indicates that the summary message of the status byte register is set. So there is information inside.
If requested via a serial poll (only with IEEE488, see chapter 2.8, 'Reading from the interface'), the RQS (request service) bit is returned. It indicates that the instrument has information to be requested (IEEE definition).

Bit 7: The set OPER (operation status) indicates that the summary message of the operation status register is set. So there is information inside (IEEE definition).

Each of the 8 bits has a value. If you add the values of all set bits you get the value of the status byte register:

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

Bit number	7	6	5	4	3	2	1	0
Bit name	OPER	MSS / RQS	ESB	MAV	QUES	QYE	--	--
Bit value	128	64	32	16	8	4	2	1

Example
You request the status byte register via a serial poll and get the number 24 (as decimal number). If you convert it you get as binary: 24d=00011000b

You see bit 3 (QUES) and bit 4 (MAV) are set.

The status byte register is fed by queues and registers which are explained in the following chapters.

3.3.2 Queues

There are two queues: the output queue and the error/event queue. Both are constructed as FIFOs.

If there is any information available (usually after a request command with a '?'), it is stored in the output queue. Then the MAV bit (bit 4 in the status byte register) is set. With RS232 interface the data of the output queue are now sent automatically. With IEEE488 the status byte should be checked via serial poll until this bit is set and then a read command from the instrument should be performed.

In the error/event queue only error messages are stored. They are generated, if you for example try to read data with a /nquery/ command or a wrong command syntax. If the message is inside the queue, the QYE bit (bit 2 in the status byte register) is set.
If your program detects that there are any error messages available you should read them with the proper commands (e.g. SCPI command :SYSTem:ERRor:ALL?). There is space for 16 error messages in the queue. If more messages exist, the 16th entry will change the error message 'Queue overflow'.

3.3.3 General construction of a register structure

All so called register are in fact register structures build up of several sub registers with special functions. This special function exist for most registers. Figure 14 shows the general set-up as described in IEEE488.2 standard.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

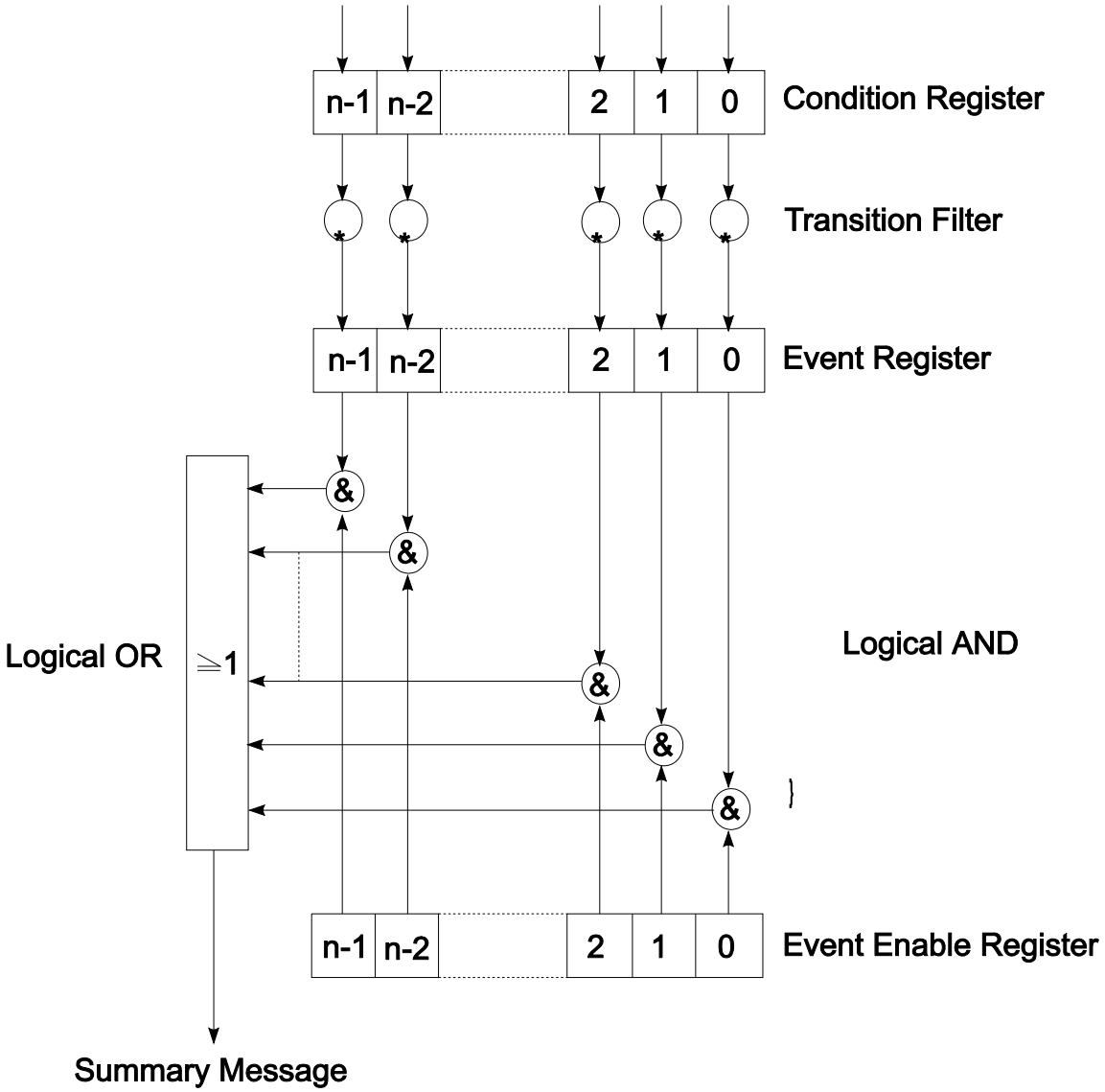

Figure 14: Content of a register structure

3.3.3.1 Condition registers

The condition registers represent the actual state of the instrument. Due to this they are read only and are also not cleared after being read. If this state was changed in the past, this cannot be seen.

3.3.3.2 Transition filters

They are not implemented in all registers. The LMG instruments contain transition filters in the operation status data structure and the questionable status register structure.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

There are two registers which can be changed independently. The transition filters are edge triggered. So they switch at a 0-to-1 transition (positive transition filter) or 1-to-0 transition (negative transition filter). If they detect 'their' edge, they set the corresponding bit in the event register (see 3.3.3.3, 'Event register')

Controlled by the transition filter you can control, which changes in the condition register are stored in the event register. If you activate both filters, all changes are stored in the event register. The transition filter can only set bits, not clear them.
If both filters are deactivated, no events will be stored in the event register.

3.3.3.3 Event register

The event register is connected via the transition filter to the condition register. So you can read, if any condition has ever appeared in the instrument. It can, but does not need to represent the current state!

The bits in the event register remain set until they are read. It is a sticky register. The register is read only.

3.3.3.4 Event enable register and summary message

The event enable register is connected via a logical AND function with the event register. Depending on the mask in the event enable registers it is possible to generate the summary message only from some defined events via the OR function. The summary message is sent to another register, e.g. the status byte register.

3.3.3.5 Example

In the negative transition filter only bit 2 is set. In the event enable register bit 2 is also set, as well as in the condition register. The event register is cleared. The summary message is assumed to be zero.

If bit 2 in the condition register then changes from 1 to 0, the negative transition filter recognises this and sets bit 2 in the event register. This is AND-combined with the event enable register. The final OR function then forces the summary message to be 1.

3.3.4 Main structures

There are three main structures. They are constructed like written above except the standard event status register.

3.3.4.1 Operation status data structure

This structure indicates, what the current actions of the instrument are. The summary message is bit 7 (OPER) of the status byte register.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

3.3.4.2 Questionable status register structure

This structure indicates what the current measured values of the instrument are. The summary message is bit 3 (QUES) of the status byte register.

3.3.4.3 Standard event status register

This structure indicates basic instrument conditions. This register has no condition register and no transition filter. The states are directly stored in the standard event status register and cleared after a read.

Bit 0 **OPC** Operation-Complete-Bit, indicating that all commands before the „*OPC“ command have been executed. This bit will only be set, when the *OPC command has been received by the LMG.

Bit 1 **RQC** Request-Control-Bit, never used by the LMGs, because this device will never become an active controller in the GPIB bus system.

Bit 2 **QYE** Query-Error-Bit, indicating that an attempt is being made to read data from the Output Queue when no output is either present or pending, or any data in the output queue has been lost.

Bit 3 **DDE** Device-Specific Error-Bit, indicating that the detected error is neither a Command Error, a Query Error, nor an Execution Error.

Bit 4 **EXE** Execution-Error-Bit. It indicates that a <Program Data> element following a header was evaluated by the device as outside of its legal input range, or a valid program message could not be properly executed.

Bit 5 **CME** Command-Error-Bit. It is used to indicate errors detected by the parser while examining the incoming commands.

Bit 6 Unused

Bit 7 **PON** Power-On-Bit. It indicates an off-to-on transition in the devices power supply.

3.4 FAQ

3.4.1 How to find a command?

In the user manuals of the LMG instruments you find two indices: One for general purposes and one for the interface commands. Here you find all topics as well as all commands.

3.4.2 How to specify different channels?

See 2.2.4, ‘Channel number (suffix)’

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

3.4.3 How to change the analogue outputs?

Here we describe how the analogue outputs of the processing signal interface are programmed.

To change the full scale of the analogue output you use the commands

SCPI: :SOURce:VOLTage:SCALe:FScale <NRf>
SHORT: AOHI <NRf>

For the zero value you use:

SCPI: :SOURce:VOLTage:SCALe:ZERO <NRf>
SHORT: AOLO <NRf>

With

SCPI: :SOURce:VOLTage:VALue <string>
SHORT: AOIX <string>

You set the value which should be output. The identifier (ID, see 2.2.1, 'Identification (ID)') is passed as a string (see chapter 2.2.8.3, '<string>').

The following example enables the output of the DC voltage of channel 3 on analogue output 2 (see 'Example 2: Reading the DC voltage'):

SCPI: :SOURce:VOLTage:VALue2 "Udc:3"
SHORT: AOIX2 "Udc:3"

3.4.4 How to get timestamps with accuracy of one microsecond from the LMG

There are two undocumented interface commands, which are used in the ZES software to get an accurate timestamp for the end of a measuring cycle. The timestamp is composed of the commands *TST? 20 and *TST? 21.

Each of these returns a 32-bit value, combined they form a 64-bit integer timestamp in microseconds. *TST? 20 represents the 32 high-order bits in the 64-bit value. The timestamp is essentially in Unix-format, but it counts the microseconds (not the seconds) since 1.1.1970. The instrument does not support different timezones, so the timestamp must be interpreted as UTC regardless of the local time to which the clock is set.

3.4.5 General hints

- Check if the cable is plugged in properly.
  Some manufacturers use connectors at their instruments which are not according to the standard. This can cause contact problems.

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

- If you are not sure how to use a command or how the response looks like, use a simple terminal program (e.g. Hyper Terminal from Windows) to test it via the RS232 interface.

3.4.6 Hardware RS232 communication logging

If you want to watch the communications that really happens on your RS232 interface, a software logging (writing to a log routine in your read() and write() functions) is not the best choice, because it shows not the real physical traffic, but only the one the computer sees at this point. You cannot see if lower level routines add or remove something. But the instrument is only interested in the data on the physical wires, so it is best to watch these.

With the circuit below you can use a second PC (or another RS232 port on the same PC) together with a terminal program to log the physical data from the PC (master) or LMG (slave). Furthermore you can watch both data streams as long as they do not send at the same time which is not usual for this application. By this you also get the actual sequence of the communication (which answer came after which request).

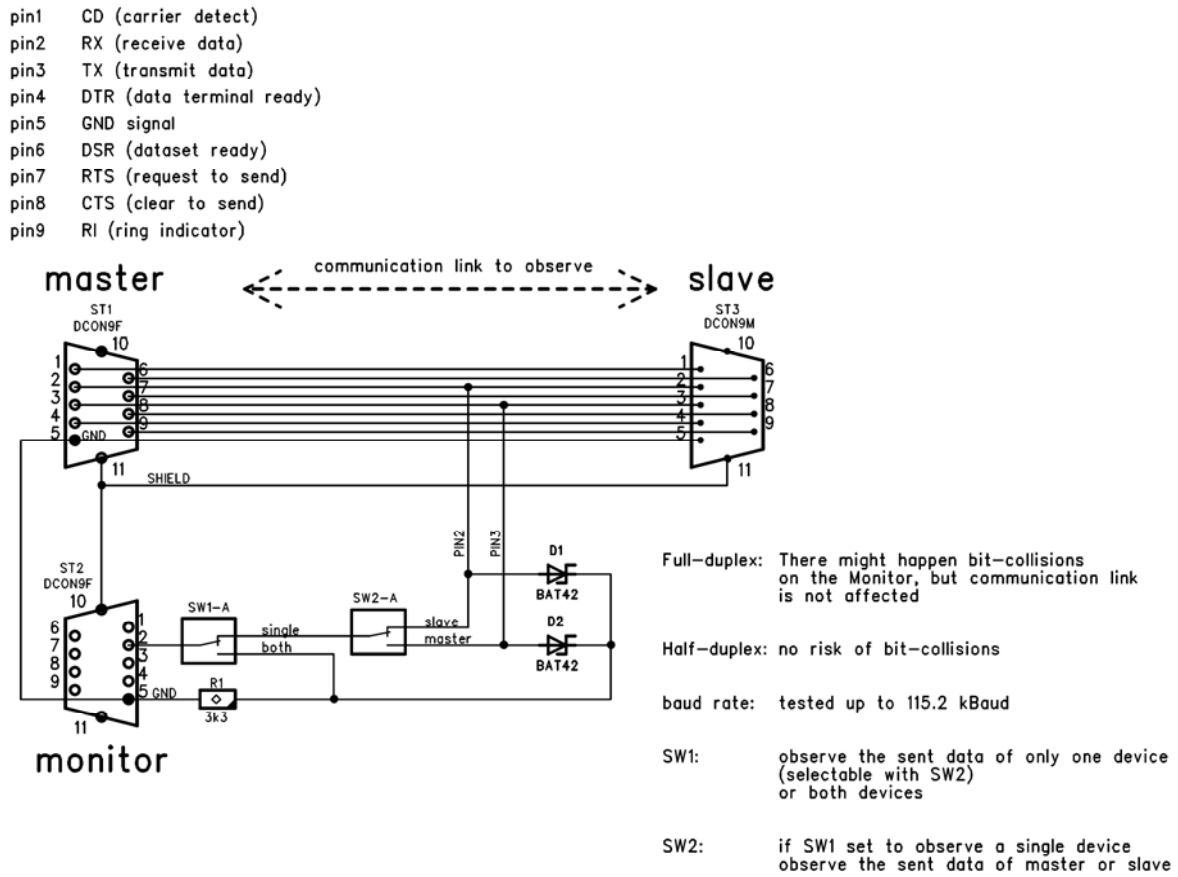

Figure 155: Circuit diagram for an RS232 observer

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com

ZES ZIMMER Electronic Systems GmbH Tel. +49 (0) 6171/6287-50 sales@zes.com www.zes.com
ZES ZIMMER Inc. Ph. + 1 760 550 9371 usa@zes.com