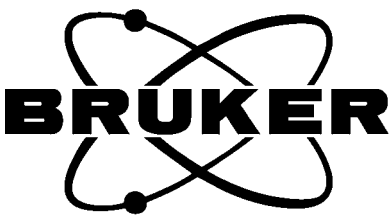
OPUS

Spectroscopy Software

Version 6

Manual

OEM DLL



1005458

This manual is the original documentation for the OPUS / OEM DLL.

Table of Contents

1 How to use the OEM DLL 1-1

2 Prerequisities 2-1

3 Interface Functions 3-1

3.1 Calling OPUS Functions 3-2

3.1.1 Calling from the Command Line 3-2

4 The DLL and a full OPUS Version 4-1

5 Implementation 5-1

5.1 Using C/C++ 5-1

5.2 Changing measurement parameters 5-2

5.3 Doing Evaluations 5-2

5.4 Using VB 5-3

6 Required Files 6-1

7 OPUS Command Reference 7-1

7.1 Command Syntax of OPUS Functions 7-1

7.2 Including OPUS Commands in Macros 7-1

7.3 Measurement Commands 7-4

7.4 Reference Section 7-5

7.5 OPUS Functions Sorted Alphabetically 7-5

7.6 OPUS Functions Sorted by Type 7-7

7.7 OPUS Manipulation Functions 7-8

7.7.1 ABTR 7-8

7.7.2 Average 7-9

7.7.3 Baseline 7-10

7.7.4 BlackBody 7-10

7.7.5 Convert 7-10

7.7.6 Cut 7-11

7.7.7 Deconvolution 7-11

7.7.8 Derivative 7-11

7.7.9 Extrapolation 7-12

7.7.10 FFT 7-12

7.7.11 FreqCalibration 7-14

7.7.12 InverseFT 7-14

7.7.13 KramersKronig 7-15

7.7.14 MakeCompatible 7-15

7.7.15 Merge 7-16

7.7.16 Normalize 7-16

7.7.17 PostFTZerofill 7-16
7.7.18 RamanCorrection 7-17
7.7.19 Smooth 7-17
7.7.20 StraightLine 7-17
7.7.21 Subtract 7-18
7.8 OPUS Evaluation Functions 7-18
7.8.1 Integrate 7-18
7.8.2 PeakPick 7-19
7.8.3 SignalToNoise 7-20
7.9 OPUS File Functions 7-20
7.9.1 ChangeDataBlockType 7-20
7.9.2 CopyDataBlock 7-20
7.9.3 DeleteDataBlock 7-21
7.9.4 Restore 7-21
7.9.5 Save, SaveAs 7-21
7.9.6 SendFile 7-22
7.9.7 Unload 7-22
7.10 OPUS Measurement Functions 7-23
7.10.1 Measurement Commands 7-23
7.10.2 SendCommand 7-23
7.10.3 SaveReference 7-24
7.10.4 LoadReference 7-24
7.11 OPUS Library Functions 7-24
7.11.1 LibrarySearchInfo 7-24
7.11.2 LibrarySearchPeak 7-25
7.11.3 LibrarySearchStructure 7-26
7.11.4 LibrarySearchSpectrum 7-27
7.11.5 LibraryInitialize 7-28
7.11.6 LibraryStore 7-29
7.11.7 LibraryEdit 7-29
7.11.8 InfoInput 7-30
7.12 Miscellaneous OPUS Functions 7-31
7.12.1 ExternalProgram 7-31
7.12.2 ParameterEditor 7-33
7.12.3 Plot 7-34
7.12.4 VBScript 7-35

8 Client/Server Reference 8-1

8.1 Overview of Available Functions 8-1
8.2 Commands and Command Syntax 8-1
8.3 Old C/S Commands 8-2
8.3.1 Overview 8-2
8.3.2 CLOSE_PIPE 8-3
8.3.3 COUNT_ENTRIES 8-3
8.3.4 READ_FROM_ENTRY 8-4
8.3.5 WRITE_TO_ENTRY 8-4
8.3.6 READ_FROM_FILE 8-5
8.3.7 WRITE_TO_FILE 8-6
8.3.8 READ_FROM_BLOCK 8-7

8.3.9 WRITE_TO_BLOCK 8-8
8.3.10 ASCII 8-8
8.3.11 BINARY 8-9
8.3.12 DATA_VALUES 8-9
8.3.13 DATA_POINTS 8-9
8.3.14 READ_HEADER 8-10
8.3.15 READ_DATA 8-11
8.3.16 WRITE_HEADER 8-13
8.3.17 WRITE_DATA 8-14
8.3.18 COPY_DATA 8-15
8.3.19 LOAD_FILE 8-16
8.3.20 UNLOAD_FILE 8-17
8.3.21 START_MACRO 8-17
8.3.22 FILE_PARAMETERS 8-19
8.3.23 OPUS_PARAMETERS 8-20
8.3.24 READ_PARAMETER 8-20
8.3.25 WRITE_PARAMETER 8-21
8.3.26 RUN_MACRO 8-21
8.3.27 MACRO_RESULTS 8-22
8.3.28 KILL_MACRO 8-23
8.4 Obsolete Commands 8-23
8.4.1 OVERWRITE 8-24
8.4.2 PRESERVE 8-24
8.4.3 TIMEOUT 8-25
8.5 New Commands 8-25
8.5.1 BYTE_MODE 8-26
8.5.2 INT_MODE 8-26
8.5.3 FLOAT_MODE 8-26
8.5.4 DOUBLE_MODE 8-27
8.5.5 HEXSTRING_MODE 8-27
8.5.6 FLOATCONV_MODE 8-27
8.5.7 GET_DISPLAY 8-28
8.5.8 SET_WINDOW 8-28
8.5.9 NEW_WINDOW 8-29
8.5.10 CLOSE_WINDOW 8-29
8.5.11 POSITION_WINDOW 8-30
8.5.12 GET_LANGUAGE 8-30
8.5.13 GET_OPUSPATH 8-31
8.5.14 GET_BASEPATH 8-31
8.5.15 GET_DATAPATH 8-31
8.5.16 GET_WORKPATH 8-32
8.5.17 GET_USERNAME 8-32
8.5.18 GET_BENCH 8-33
8.5.19 UPDATE_BENCH 8-33
8.5.20 COMMAND_SAY 8-34
8.5.21 REPORT_INFO 8-34
8.5.22 HEADER_INFO 8-35
8.5.23 MATRIX_INFO 8-36
8.5.24 MATRIX_ELEMENT 8-36
8.5.25 HEADER_ELEMENT 8-37

8.5.26 COMMAND_MODE 8-38
8.5.27 EXECUTE_MODE 8-39
8.5.28 REQUEST_MODE 8-39
8.5.29 CLOSE_OPUS 8-40
8.5.30 TAKE_REFERENCE 8-40
8.5.31 MEASURE_SAMPLE 8-40
8.5.32 COMMAND_LINE 8-41
8.5.33 STOP_THREAD 8-42
8.5.34 ACTIVATE_DIALOG 8-42
8.5.35 LOAD_EXPERIMENT 8-43
8.5.36 GET_USERRIGHTS 8-43
8.5.37 PACKET_AVAILABLE 8-44
8.5.38 GET_CLIENTAREA 8-44
8.5.39 ACTIVATE_DISPLAY 8-45
8.5.40 GET_LIMITS 8-45
8.5.41 SET_LIMITS 8-46
8.5.42 DISPLAY_BLOCK 8-46
8.5.43 UNDISPLAY_BLOCK 8-47
8.5.44 ENUM_STRINGS 8-47
8.5.45 GET_VERSION 8-48
8.5.46 ASK_THREAD 8-49
8.5.47 FIND_FUNCTION 8-49
8.5.48 WORKBOOK_MODE 8-50
8.5.49 GET_SELECTED 8-50
8.5.50 LIST_BLOCKS 8-51
8.5.51 SHOW_TOOLBAR 8-52
8.5.52 HIDE_TOOLBAR 8-52
8.5.53 QUICK_PRINT 8-53

9 Problems and Solutions 9-1

9.1 Startup problem 9-1
9.2 "Failed to open document" 9-1
9.3 File not found in filetable 9-1
9.4 Error in commandline 9-1
9.5 VB error 48/53 9-2

1 How to use the OEM DLL

For OEM customers or system integrators, we have a special option to get data from our instruments.

It consists of a main Opus-DLL with a few support DLLs for FT and data handling etc..

This will allow communication to instruments and doing measurements as well as basic data manipulation and evaluation functions.

In case special packages like QUANT are needed an OEM would be able to simply add a few more DLLs to the distribution and receive new registration codes for the extended packages

Therefore if ALL these DLLs containing Opus evaluation or manipulation functions are packed onto the developers CD, and it is thus possible to perform nearly every function available in a full Opus.

Functions of the DLL are directly called from the custom program (C calling conventions).

An OEM customer will be supplied with a CD containing:

- - the necessary DLLs,
- - a header file with the function declarations,
- - a library to link with his application and
- - some sample projects.

The code was written and tested with VC6/Visual Studio 2003 and VB.Net.

In addition, customers have successfully used the DLL from:

- C++ VC6 + Borland + VS2003
- Visual Basic
- VB.NET
- Java and
- Delphi

The environment being XP, XP embedded or Windows 2000.

2 Prerequisities

Registration needed

To use the DLLs, an OEM must make sure that:

- the customer owns a Bruker Optik spectrometer,
- the customer owns legally a license (called OPUS/DLL) to use the OPUS-DLLs on this spectrometer and
- the Bruker Optik end-user license terms are valid/observed.

Spectrometers will have to be equipped minimum with the basic OPUS package (OPUS/IR) in order to facilitate for service and/or validation.

If a full OPUS version is also installed enter the registration data in Opus and it will work for the DLL as well.

In this case both should be of the same version i.e. OPUS6 with DLLs version 5 might cause problems and vice versa.

Access

In addition you need write access to the current application directory for some temporary files. Furthermore, the .ows file also defines where the intermidiate work files will be stored what might cause access problems as well. In this case try replacing the .ows file with the original version from the CD.

3 Interface Functions

Functions that can be called via the DLL interfaces are only five including:

1) Initialization to connect to the instrument
2) Scan background command
3) Measure Sample command
4) GetCollectedData command plus
5) the full set of client/server commands know from DDE/Pipe/Script to fulfill ANY other needs.

This should be a very simple way to acquire data. However, basic knowledge of the normal operation of the instrument under Opus will be needed.

The prototypes of the available functions as in the file BO_OEM.H are as follows:

```
UINT InitInstrument(LPCTSTR pConfigFile, LPTSTR pResult, UINT
uiBufferLen);
UINT BackgroundScan(LPCTSTR pXPM, LPTSTR pResult, UINT
uiBufferLen);
UINT SampleScan(LPCTSTR pXPM, LPTSTR pResult, UINT uiBufferLen);
UINT GetAcquiredData(LPTSTR pResult, UINT uiBufferLen);
UINT DirectCommandCall(LPCTSTR pCommand, LPTSTR pResult, UINT
uiBufferLen);
```

Each function takes a parameter pResult that is a Pointer to a buffer that will receive the result of the function and a parameter uiBufferLen that holds the length of the buffer that receives the result.

All the pResults parameters are [Out], i.e. the result will be written into that buffer. The rest is always [In].

Each function will directly return the length of the result. The result usually is "OK" or an error message or the measured data written into pResult

InitInstrument connects to an Instrument. The instrument is defined by an .NTI file found in the same directory as the application. The name is passed in pConfigFile. Call this function once when the application is initializing.

The .nti file describes the type of the optical bench. So you have to choose the one corresponding to your instrument,

"Reconnect request started" would be a correct answer to the InitInstrument call. So this did work.

BackgroundScan executes a background measurement. The parameters are defined by an Opus Experiment file whose name is passed in pXPM.

OK0 indicates that the background measurement has been completed.

SampleScan executes a sample measurement. The parameters are defined by an Opus Experiment file whose name is passed in pXPM.

GetAcquiredData receives the data collected by the last SampleScan command.

DirectCommandCall allows the execution of any Opus Client/Server command, see reference chapters for details. A command line is send as text and for Opus functions it will look like:

3.1 Calling OPUS Functions

Typical OPUS manipulation functions can also be called as a text command. The corresponding uniform syntax can be used wherever a function specification is started by a text command in OPUS:

- in the command line interface
- within macros
- in DDE requests of an external client program to the OPUS server
- within scripts

Syntax:

<Function> ([File],{Parameter})

This chapter gives a brief overview. For details on each calling option refer to the following chapters.

3.1.1 Calling a Command Line

OPUS contains a simple command line interpreter.

For example, if you enter

```
Baseline (["e:\opus\data\abboe05.0"],{})
```

aa the command line, the file ABBOE05.0 will be baseline corrected. If the spectrum file has not been loaded in OPUS, it will be automatically loaded prior to the baseline correction.

The file (including path) to be processed has to be enclosed in inverted commas. The empty braces at the end of the command indicate that no parameters have been specified. In this case the default values will be used. However, if you want to make specific settings, you have to enter the corresponding OPUS parameters. In general, OPUS parameters consist of a three-character code, followed by an equal sign and a value (e.g. BME=1). Several parameters have to be separated by a comma. Which parameters you have to use with which values depend on the particular functions. For a detailed description, see the respective function.

A file can also be loaded more than once at the same time. Therefore, a number is added after the file name to identify the version of the file. This number is called clone count. Furthermore, a data file can consist of several data blocks, which can be addressed separately by a colon:

```
Baseline (["e:\opus\data\abboe05.0" 3:AB], {})
```

The above command processes the absorption data block (AB) of the third version (copy) of the ABBOE05.0 file.

Despite Opus using multitasking heavily inside this DLL functions will only return once the command has been completed. However, you can call them from a background thread but make sure that your program does not multiply call these functions at the same time!

4 The DLL and a full OPUS Version

Both OPUS and an application using the DLLs can be used on the same PC. However, only one can run at each time.

We do register the application to prevent two instances running at the same time which might cause all sorts of problems. E.g. when multiple instances try to access the same optical bench. You will get a message that reads "Sorry we are already running" (or sometimes an empty box since the resource is not loaded until InitInstrument) if you did try running the second instance incidentally.

There is one thing that might interfere here that is the DLL will share the last workspace used by Opus itself. After using OPUS take a look in the registry under:

HKEY_CURRENT_USER\Software\BRUKER\OPUS\Recent File List

and make sure that this points to a valid workspace as well or remove that particular key completely to use the DLL.

Also both use the same location in the registry to store the information about the last instrument used. Therefore, check what is registered under:

HKEY_LOCAL_MACHINE\SOFTWARE\BRUKER\OPUS\HARD-WARE\Bench

resp. restore the correct value from within your program.

5 Implementation

When implementing your application you can either statically link to the Opus.dll or use LoadLibrary. Statically linking is recommended but LoadLibrary does work as well. In that case do not unload the DLL in your program until you exit. The point is that the mutex used to prevent two instances is only freed when the complete process i.e. your program terminates.

You don not have to register the opus.dll. Functions inside the DLL are directly called from your application using C calling conventions

In case LoadLibrary returns null this does mean an error occured during loading or initialisation. One reason might be that some files (other dlls) are missing. Make sure that your application and all the files from the CD are in the same directory.

Inside the DLL some decorated names are used. Usually you won't see these names if you use the bo_oem.h include file and the library opus.lib which also comes with the CD.

5.1 Using C/C++

To compile an OEM application, you need from the CD the header file

BO_OEM.h

And to link the application you need the library file

OPUS.LIB

A simple VC6 sample is included on the CD called CALLDLL

All function calls are found in the module calldll.cpp and are basically always done in the same manner:

Reserve space for answer, the call function and show the result:

```
char Buffer[1024];
InitInstrument("matrix.nti",Buffer,1024);
AfxMessageBox(Buffer);
```

After initializing the instrument you can use the other function calls like a sequence BackgroundScan, SampleScan and GetAcquiredData to get the resulting spectrum. See sample code on the CD.

5.2 Changing measurement parameters

For the simple sample above all measurement parameters are defined by the experiment (.XPM) file. If you must change several of the parameters either define several sets of .XPMs or overwrite them in the commands.

Directly modifying the XPM is NOT the way to go. This is a binary format that you really don't want to access.

If you have to change parameters like resolution or number of scans we cannot go with the simple DLL function calls but rather send some more direct commands to Opus that allow you to change all the parameters. This is done via a call to "DirectCommandCall" instead of SampleScan. As the command send a string that is formatted like this:

"MeasureSample (0, {EXP='test.XPM', XPP='C:\OPUS\XPM', NSS=123})"

EXP defines the basic experiment file, XPP the corresponding path and NSS sets the number of scans to 123. This command will return the result filename and you will have to send a few more commands to get the actual data:

READ_FROM_FILE ResultFile
READ_FROM_BLOCK AB (if you acquire absorbance data)
DATA_POINTS
READ_DATA (get the results as text)
UNLOAD_FILE ResultFile (for cleanup)

5.3 Doing Evaluations

A more complicated sequence would e.g. include some parameter overwriting and maybe the call to a Quant method for an evaluation.

The following sequence demonstrates the basic principle how this is going to be achieved.

```
                CString csXPM("Simulator.xpm"), csQ2("test.q2");
                char Buffer[1024];
                CString csCommand = "MeasureSample({WRK=3, MOP='EXP
XPP DEL', EXP='";
                csCommand += csXPM;
                csCommand +="', XPP='d:\\oem55', DEL=0})";
                DirectCommandCall(csCommand,Buffer,1024);
                CString csResult = CString((char*)Buffer);
                INT pos1 = csResult.Find('\n');
                INT pos2 = csResult.Find('\n', pos1+1);
                pos1 = csResult.Find('\n', pos2+1);
                pos2 = csResult.Find('\n', pos1+1);
                CString csName = csResult.Mid(pos1+1, pos2-pos1-1);
```

```
                csCommand = "Quant2 ([";
                csCommand +=csName;
                csCommand +=":AB], {QP2='D:\\oem55', QF2='";
                csCommand += csQ2;
                csCommand +="'});";
                DirectCommandCall(csCommand,Buffer,1024);

                csCommand = "READ_FROM_FILE ";
                csCommand +=csName;
                DirectCommandCall(csCommand,Buffer,1024);
                DirectCommandCall("READ_FROM_BLOCK AB/
Quant",Buffer,1024);
                DirectCommandCall("MATRIX_ELEMENT 1 0 1
2",Buffer,1024);
                csResult = CString((char*)Buffer);
                CString csQuantResult = csResult.Right(csResult.GetLength()-
4);

                csCommand = "Unload ([";
                csCommand +=csName;
                csCommand +=":AB], {});";
                DirectCommandCall(csCommand,Buffer,1024);
```

The OPUS DLL supports also multiplexed Bruker instruments with more than one probe. You can use the "CHN=Fibre 2" etc. commands to switch between the different probes.

These instruments acquire data for one probe at a time. Thus you have to call the measurement for each channel.

5.4 Using VB

We implemented a very simple project in VB6 that calls all the necessary interface functions of the DLL that can be found on the CD.

You don't need any header or .lib files, the functions just have to get introduced to VB using "Declare function":

```
'Functions exported from OPUS.DLL sorted by ordinal number:
 Public Declare Function BackgroundScan Lib "Opus.dll" _
                Alias "#1" _
                 (ByVal XpmFile As String, _
                  ByVal Result As String, _
                  ByVal BufferLen As Integer) As Integer
 Public Declare Function DirectCommandCall Lib "Opus.dll" _
                Alias "#2" _
                 (ByVal Command As String, _
                  ByVal Result As String, _
                  ByVal BufferLen As Integer) As Integer
 ' ForRunDLL32 ordinal number 3
 Public Declare Function GetAcquiredData Lib "Opus.dll" _
                Alias "#4" _
```

```
                    (ByVal Result As String, _
                     ByVal BufferLen As Integer) As Integer
Public Declare Function InitInstrument Lib "Opus.dll" _
                   Alias "#5" _
                    (ByVal ConfigFile As String, _
                     ByVal Result As String, _
                     ByVal BufferLen As Integer) As Integer
Public Declare Function SampleScan Lib "Opus.dll" _
                   Alias "#6" _
                    (ByVal XpmFile As String, _
                     ByVal Result As String, _
                     ByVal BufferLen As Integer) As Integer
```

In the Sub Form_Load() you would then e.g. load the library and call InitInstrument:

```
Dim strResult As String
Dim RetVal As Integer
Dim Buffer As String
Dim nHMod As Long

nHMod = LoadLibrary("opus.dll")

Buffer = String(1024, vbNullChar)
RetVal = InitInstrument("matrix.nti", Buffer, 1023)
strResult = Buffer
Text1.Text = strResult
```

Issuing command would be done as follows:

```
Private Sub Command1_Click()
   Dim strResult As String
   Dim intResult As Integer
   Dim Buffer As String
   Buffer = String(1024, vbNullChar)
   intResult = DirectCommandCall("GET_VERSION", Buffer, 1024)
   strResult = Buffer
   Text1.Text = strResult
End Sub
```

The GET_VERSION command of this sample will return the version of the currently used DLL.

6 Required Files

There are many files on the CD-ROM, because we offer different DLLs providing a lot of different functions also for OEM applications.

Only a few of those are needed if you are just measuring spectra.

Even if the DLL has only 5 function calls it is possible via the DirectCommandCall function to do almost any evaluation or manipulation that is possible within a normal Opus. E.g. a quantitative analysis, but if you do not do this you will not have to include the quant.dll.

You might e.g. need instrument tests etc. and those do require a lot of function apart from measuring and doing an FT. E.g. Integration, reading Reports, Smooth, Spectrum Calculator, PeakPicking, SignalToNoise etc.

To redistribute an application, you need the following DLL in the same directory as the main program (or somewhere in the search path):

- opus.dll
- fileutil.dll
- measure.dll
- opusfft.dll
- macro.dll
- display.dll
- fileutil.dll
- nwc.dll
- nwutil.dll
- og70as.dll
- onldisp.dll
- opusutil.dll
- osc60as.dll
- otp60as.dll
- pledll.dll
- plot3d.dll
- wct32dr3.dll
- wrt32dr3.dll
- wtOPCSvr.dll
- wtClient.dll

Additionally the following parameter files are needed:

- parmtext.bin
- original.p
- opus.par
- opus.fnc

And at least one *.nti file defining the instrument and an *.XPM file defining the measurement parameters.

If plotting is desired add also the *.PLE files.

Simple evaluation and manipulation function are found in these DLLs which will be needed for operation e.g. like OVP.

- evaluate.dll
- interactive.dll
- manipulate.dll

The rest of the DLL is only needed when special packages like Quant etc. should be used via the Opus.dll:

- birsy.dll
- quant.dll
- quanta.dll
- ident.dll
- semi.dll
- file3d.dll
- procctrl.dll
- heppfrie.dll
- neuro.dll
- manipulate_2.dll

The workspace: e.g. default.ows

All the recent parameter settings for all opus functions are stored in the workspace (.ows). With the DLL you will most likely define the parameters yourself. Therefore you only have to supply the default setting.

However, the DLL will check the registry for the MRU .ows file and use that with precedence. If none is registered default.ows will be used.

The userdatabse: userdatabase.dat

If the is a userdatabe from an Opus version this will be used by the dll. But normally this will not be present and the dll will instead create an intermediate one.

The instrument definition: e.g. matrix.nti

What .nti file should I use in the InitInstrument function?

That depends on the instrument you want to talk to. If the kind of optical bench you are using is same kind of Matrix matrix.nti would be a good starting point. If you have Opus running with the instrument check the "Optical Bench" Tab under "Optic Setup and Service" and use the name you see in the Configuration field. You don't have to ship the other .nti files as well.

The NTI files must exist. Opus will update these files whenever the instrument configuration changes.

For the NTI files this is not necessary to touch them since the optical bench will automatically tell Opus to update the contents when e.g. a new detector is plugged in.

The ONLY change you might consider is changing the IP address. The IP address of the instrument is stored on the instrument and it has to be predefined in advance for Opus inside the .NTI file.

The NTI file only defines what type of instrument is available and where (IP) it is found. If you modify the address in the optics make sure you change the NTI file as well.

The measurement parameters: *.xpm

The XPM files do contain all the measurement parameters. This is an OPUS specific binary format and the only reasonable way to create these files is via OPUS. Use an instrument that is setup with the settings as you need them and let Opus store this setup in an XPM file.

XPMs should not be generated by direct commands, because normally you use an existing one and can overwrite any parameter directly in the command.

That is the XPM defines only the default values and everything that is send within the command line will overwrite the value from the XPM file.

Is there a limitation to "change" a value of one parameter?

No, the following ones are only the most common parameters. Usually people using these commands don't modify more than these:

SNN, SFM, CNM, XPM, XPP, RES, NSS

But, you can pass any parameter that the instrument supports also in the same command line.

Make sure to use measurement parameters that are appropriate for the instrument that you are connected to. I.e. for using a Matrix instrument you need a matrix.nti file and e.g. a matching matrix.xpm. Otherwise you might get error messages complaining about illegal parameters.

7 OPUS Command Reference

The OPUS commands accessible from the OPUS pull-down menus call OPUS processing functions, that in turn perform the desired manipulation. These OPUS processing function can be included in macros, scripts and external programs. Alternatively to launching a function via the OPUS pull-down menu command, the function name can be typed in the OPUS command line.

7.1 Command Syntax of OPUS Functions

Syntax:

CommandName (Input List 1, ..., Input List n, {PAR 1=Value 1, ..., PAR n=Value n});

CommandName	name of the OPUS command.
Input List n	list of input files (see below).
PAR n	three letter parameter name n.
Value n	value for parameter n.

Syntax for file list:

([<File 1>:BlockID 1] ... [<File n>:BlockID n]

*<File n>*name of input file or the file variable n.

*BlockID n*name of the data block of file n.

Note that the files in a list are separated by blanks, while the lists themselves are separated by commas. Most functions require only one file list; a few files however, (like *Make Compatible* or *Subtraction*) need several file lists.

7.2 Including OPUS Commands in Macros

We strongly recommend to only use the Macro Editor, if you want to include OPUS commands into macros. Using the Macro Editor guarantees that all relevant parameters required by the command are inlcuded in the command line and PARAMETER section. Furthermore, it is ensured that these parameters are initialized with valid values.

To append an OPUS command to a macro, simply select the command from the OPUS pull-down menu, while the Macro Editor is running. Choose the appropriate parameters, files and settings as usual in the dialog box of the command.

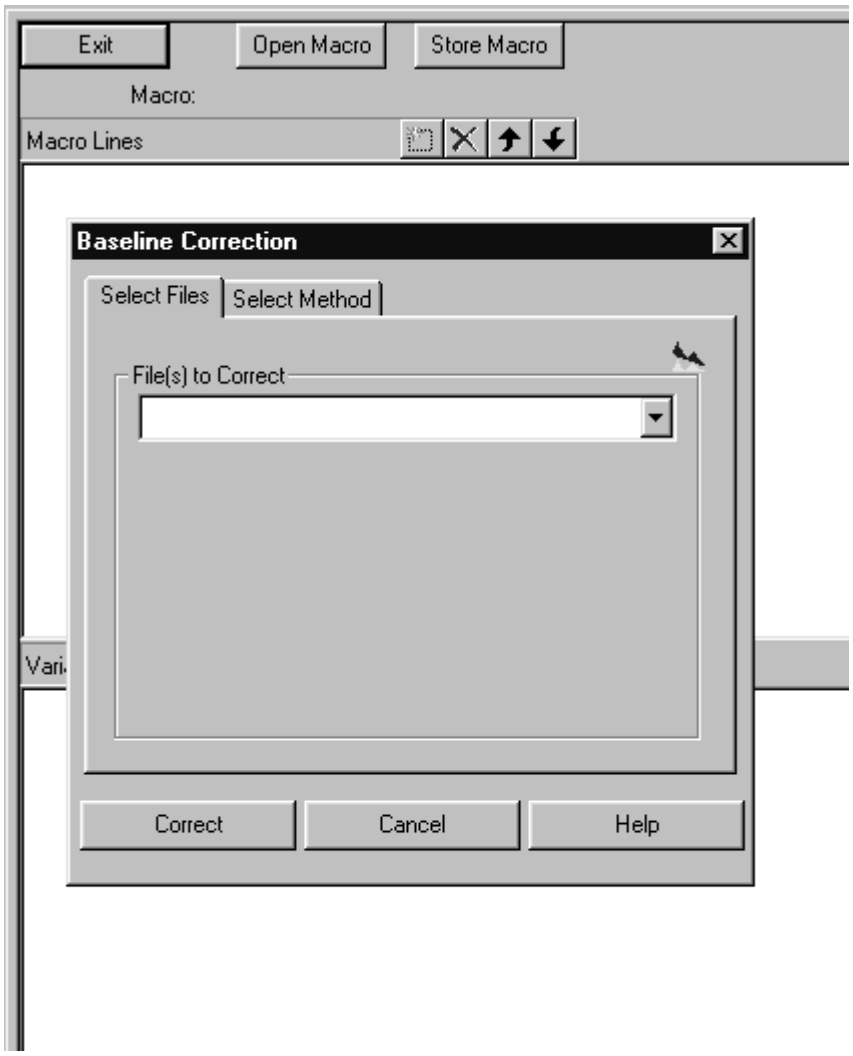


Figure 1: Including an OPUS Command

After clicking on the *Execution* button in the dialog box for executing the command, the OPUS command dialog box is replaced by a parameter dialog, that lists all parameters relevant for the processing function. When you click the *OK* button, the respective OPUS processing function will be appended to the macro.

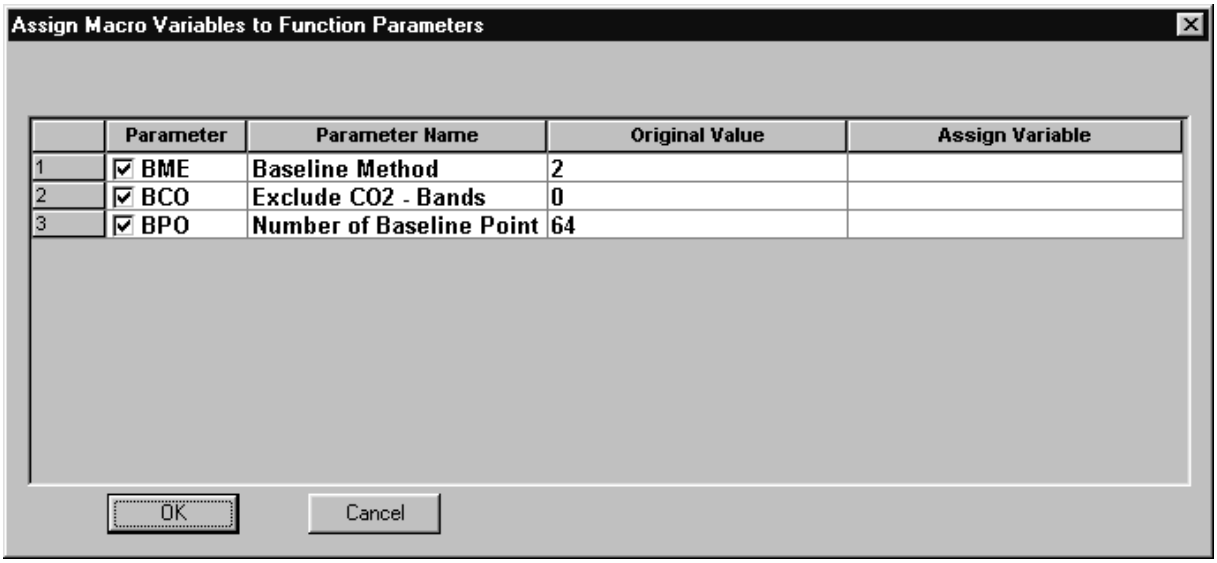


Figure 2: Including an OPUS Command – Parameter Dialgo Box

Column 1: Abbreviation of the parameter and check box

Column 2: Parameter name

Column 3: Parameter value as set in the OPUS command dialog box

Column 4: Assigned macro variable

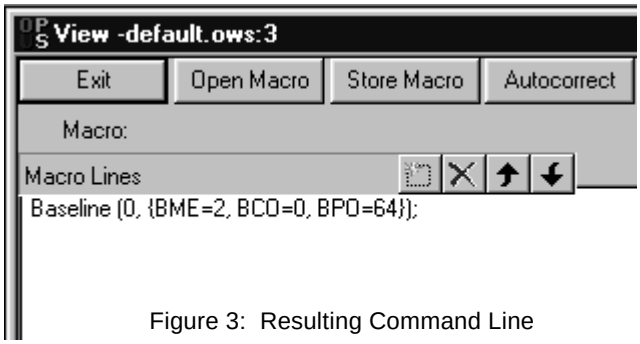


Figure 3: Resulting Command Line

Whether a parameter will be appended to the command line or included in the PARAMETER section is controlled by the check box. If the check box is not selected, a parameter entry will be made in the PARAMETER section. We recommend to always include all parameters in the command line. This ensures, that the commands are using correct parameter values at the time of command execution, in case a command or a goup of commands accessing the same parameter is repetedly used. A parameter may only appear once in the PARAMETER section and therefore, the parameter can only have one value. The only exception to this rule are the measurement commands, explained in detail in the following chapters.

Note: parameters, that have been assigned macro variables must appear in the command line!

A combobox is displayed above the parameter list of OPUS commands, which return a result or a file to the macro. From this box, you have to indicate the variable supposed to hold the returned data. Although the OPUS command will be processed correctly by the macro even if no variable was chosen, the returned data then is not accessible.

7.3 Measurement Commands

As already mentioned, the measurement commands differ from the rest of the OPUS commands. When you include the *Measurement* command in a macro, you will find that only two parameters XPP and EXP are selected by default. XPP represents the directory of the experiment file and EXP the name of the experiment. It is highly recommended to assign macro variables to these parameters. This guarantees, that a measurement started from a macro always uses an existing experiment file (and therefore a defined parameter set). For measurement functions, the remaining parameters won't be included in the PARAMETER section!

Other parameters than XPP and EXP should only be selected, if they are intended to replace values stored in the experiment file or if macro variables should be assigned to these parameters. This will become clear, if one looks at the sequence in which a measurement command is executed.

Measurement without Using an Experiment File (not recommended)

1) The measurement primarily uses the values entered in the PARAMETER section, if anything.
2) Parameter included in the command line override the values declared in the PARAMETER section.

Example:

[<File>] = MeasureSample (0, {NSS = 16});

Regardless of the original settings the measurement will now run 16 scans.

Measurement Using an Experiment File (XPP and EXP Selected)

The parameters of the PARAMETER section are ignored, and the parameters stored in the experiment file will be used instead. Again, parameters included in the command line override the values stored in the experiment file.

Example:

[<File>] = MeasureSample (0, {XPP = '<XMP Path>', EXP = 'default', NSS = 16});

Regardless of the settings stored in the experiment file, the measurement will now run 16 scans.

7.4 Reference Section

The following section describes the OPUS commands in detail. The sections are all structured in the same way. You will find:

- the title, which consists of the OPUS command referenced in this section.
- a summary of the command.
- an indication, whether the command modifies files or not.
- an explanation of the syntax.
- a table, listing all command parameters and their function.
- a note, if the command has not been implemented in OPUS to this point in time.

All of the parameters you will find listed in the tables are required, and must be stated as a part of the command. A parameter statement should therefore be included either in the parameter list of the command or in the PARAMETER section of the macro. If no parameter statement was made in a macro, OPUS will use the parameters of the active parameter set, when executing the macro. This usually leads to unpredictable results.

7.5 OPUS Functions Sorted Alphabetically

A

ABTR	absorbance transmittance conversion
Average	averages spectra

B

Baseline	performs a baseline correction of a spectrum
BlackBody	Black Body generation

C

ChangeDataBlockType	changes the type of a data block
Convert	converts spectra
CopyDataBlock	copies a data block from one file to another
Cut	cuts a frequency range out of a spectrum

D

Deconvolution	Fourier self deconvolution
DeleteDataBlock	deletes the specified data block
Derivative	calculates the derivative

E

ExternalProgram	starts an external program
Extrapolation	extrapolates spectra

F

FFT	Fast Fourier transformation
FreqCalibration	frequency calibration

I

InfoInput	adds an information block to a file
Integrate	integrates a spectrum
InverseFT	performs an inverse Fourier transformation

J

JCAMPToOPUS	converts a JCAMP-DX file to OPUS format

K

KramersKronig	performs a Kramers Kronig transformation

M

MakeCompatible	makes spectra compatible
MeasureReference	measures a background spectrum
Merge	merges spectra

N

Normalize	normalizes a spectrum

P

PeakPick	creates a peak table
Plot	plots spectra
PostFTZerofill	Post Zerofilling of a spectrum

R

RamanCorrection	applies Raman correction
Restore	restores an original data file

S

Save	saves a spectrum file
SendFile	sends a file via e-mail
SignalToNoise	calculates the Signal-to-Noise ratio

Smooth	smooths a spectrum
StraightLine	inserts a straight line in a spectrum
Subtract	subtracts one or more spectra from another spectrum

U

Unload	removes a spectrum from the Browser

7.6 OPUS Functions Sorted by Type

Manipulation Functions

ABTR	absorbance transmittance conversion
Average	averages spectra
Baseline	performs a baseline correction of a spectrum
BlackBody	Black Body generation
Convert	converts spectra
Cut	cuts a frequency range out of a spectrum
Deconvolution	Fourier self deconvolution
Derivative	calculates the derivative
Extrapolation	extrapolates spectra
FFT	Fast Fourier transformation
FreqCalibration	frequency calibration
InverseFT	performs an inverse Fourier transformation
KramersKronig	performs a Kramers Kronig transformation
MakeCompatible	makes spectra compatible
Merge	merges spectra
Normalize	normalizes a spectrum
PostFTZerofill	Post Zerofilling of a spectrum
RamanCorrection	applies Raman correction
Smooth	smooths a spectrum
StraightLine	inserts a straight line in a spectrum
Subtract	subtracts one or more spectra from another spectrum

Evaluation Functions

Integrate	integrates a spectrum
PeakPick	creates a peak table
SignalToNoise	calculates the Signal-to-Noise ratio

File Functions

ChangeDataBlockType	changes the type of a data block
CopyDataBlock	copies a data block from one file to another

DeleteDataBlock	deletes the specified data block
Restore	restores an original data file
Save	saves a spectrum file
SendFile	sends a file via e-mail
Unload	removes a spectrum from the Browser

Measurement Functions

MeasureReference	measures a background spectrum
SendCommand	send an optics command to the optics bench
SaveReference	saves a reference spectrum from the AQP to disk
LoadReference	loads a reference spectrum from the disk into the AQP

Library Functions

LibrarySearchInfo	information search in library files
LibrarySearchPeak	peak search in library files
LibrarySearchStructure	structure search in library files
LibrarySearchSpectrum	spectrum search in library files
LibraryInitialize	creates a new, empty library file
LibraryStore	stores a new entry in a library file or replaces an existing one
LibraryEdit	edits an entry, the library description, and the definition of information stored in a library file.
InfoInput	adds an information block to a file or edits an existing one

Miscellaneous Functions

ExternalProgram	starts and external program
Plot	plots spectra

7.7 OPUS Manipulation Functions

7.7.1 ABTR

Absorbance → Transmittance conversion.

This functions modifies the selected spectrum and changes the data block type accordingly.

ABTR ([<File>:BlockID] ...., {...});

Parameter	Value	Description
CCM	1	automatic
	2	AB → TR
	3	TR → AB

7.7.2 Average

Averages spectra.

This command requires three file lists:

File List 1: Spectra to be averaged.

File List 2: (optional) File to store the average result.

File List 3: (optional) File to store the standard deviation result.

If File List 2 and/or 3 are not specified, they have to be set to “0”.

Average ([<File 1>:BlockID 1], [<File 2>:BlockID 2], [<File 3>:BlockID 3], {...});;

Parameter	Value	Description
QA0	0	Do not average with number of scans
	1	Average with number of scans
QA2	0	Don’t create average report
	1	Create average report
QAE	NO	Don’t create standard deviation spectrum
	YES	Create standard deviation spectrum
QAF	NO	Don’t update standard deviation spectrum
	YES	Update standard deviation spectrum
QAL	LIS	Average selected files
	FIL	Average files selected by name and path
QAM	Text	Path of the files to be averaged
QAN	Text	Name of the files to be averaged
QAO	Numerisch	BlockID of the files to be averaged
QFB	Text	Path of the IDENT method
QFC	Text	Name of the IDENT method

7.7.3 Baseline

Performs a baseline correction of a spectrum.

This command modifies the selected spectrum.

Baseline ([<File>:BlockID] ...., {...});

Parameter	Value	Description
BME	1	Rubber Band correction
	2	Scattering correction
BPO	10 ... 200	number of baseline points
BCO	0	include CO_2 bands
	1	exclude CO_2 bands

7.7.4 BlackBody

Calculates a spectrum of a Black Body radiator.

This function adds a single channel sample data block to the selected file(s).

BlackBody ([<File>:BlockID] ...., {...});

Parameter	Value	Description
QTE	pos. number	temperature of the Black Body radiator
QPM	0	energy
	1	photons

7.7.5 Convert

Converts spectra.

This functions modifies the selected spectrum and changes the data block type accordingly.

Convert ([<File>:BlockID] ...., {...});

Parameter	Value	Description
CSD	1	AB, TR, Refl
	2	KM $\rightarrow$ Refl
	3	AB, TR $\rightarrow$ ATR
	4	ATR $\rightarrow$ AB

	5	Refl → lgRefl
	6	lgRefl → Refl
	7	ScSm → Raman
	8	Raman → ScSm

7.7.6 Cut

Cuts out a frequency range of a spectrum file.

This functions modifies the selected spectrum file.

Cut ([<File>:BlockID] ...., {...});

Parameter	Value	Description
CFX	number	X-start frequency
CLX	number	X-end frequency

7.7.7 Deconvolution

Performs a Fourier self deconvolution.

This functions modifies the selected spectrum.

Deconvolution ([<File>:BlockID] ...., {...});

Parameter	Value	Description
DSP		peak form
	LO	Lorentzian
	GA	Gaussian
DEF	pos. number	deconvolution factor
DNR	pos. number	noise reduction factor
DES	number	X-start frequency
DEE	number	X-end frequency
DWR	0	frequency limits
	1	file limits

7.7.8 Derivative

Calculates the derivative of a spectrum.

This functions appends a new data block, containing the derivative of the spectrum, to the original data.

Derivative ([<File>:BlockID] ...., {...});

Parameter	Value	Description
QSP	5, 9, 13, 17, 21, 25	number of smoothing points
QOD	1...5	order of derivative

7.7.9 Extrapolation

Extrapolates a spectrum.

This functions modifies the selected spectrum.

Extrapolation ([<File>:BlockID] ...., {...});

Parameter	Value	Function
QX0	number	extrapolate to zero
QX1	number	extrapolate to infinity
QX2	number	lower frequency limit
QX3	number	upper frequency limit
QX4	number	new end frequency

7.7.10 FFT

Performs a Fast Fourier transformation..

This command performs a fast Fourier transformation of an interferogram. The result is a single channel spectrum data block, which will be added to the file.

FFT ([<File>:BlockID] ...., {...});

Parameter	Value	Description
FTS	number	start frequency of the spectrum
FTE	number	end frequency of the spectrum
FZF	pos. number	Zerofilling factor
FTR	pos. number	resolution
FHR	pos. number	phase resolution
FBW		bit code used to indicate forward/backward or even/odd
	1	forward interferogram

	2	backward interferogram
	8	even separation
	16	odd separation
FTA		apodization function
	BX	Boxcar
	TR	Triangular
	4P	Four Point
	HG	Happ-Genzel
	B3	Blackman-Harris 3-term
	B4	Blackman-Harris 4-term
	NBW	Norton-Beer, weak
	NBM	Norton-Beer, medium
	NBS	Norton-Beer, strong
FLR	pos. number	limit resolution
FHZ		phase correction
	ML	Mertz
	SM	Signed Mertz
	PW	Power spectrum
	MS	Mertz stored phase
	NO	No – save complex data
FZF	pos. number	Zerofilling factor
FNL	0	no nonlinearity correction
	1	nonlinearity correction
FNC	pos. number	nonlinearity correction – detector cutoff
FNE	pos. number	nonlinearity correction – mod. efficiency
FSM		ZPD search mode
	AL	largest absolute value
	MN	minimum
	MX	maximum
	MI	mid position between min and max
	NO	use stored value
	MA	manual input
FPP	pos. number	peak position
FSR	pos. number	search range
FSY		symmetry for search range
	0	symmetrical
	1	antisymmetrical

	2	automatic
FTT		to do list — bit list for result data blocks
	1	absorbance
	2	interferogram
	4	single channel
	8	power spectrum
	16	phase spectrum
	64	single channel (real)
	128	single channel (imaginary)

7.7.11 FreqCalibration

Performs a frequency calibration.

This functions modifies the selected spectrum.

FreqCalibration ([<File>:BlockID] ...., {...});

Parameter	Value	Description
QF0	NO	do not restore original values
	YES	restore original values
MWC	number	factor
AWC	number	offset

7.7.12 InverseFT

Performs an inverse Fourier transformation.

This command performs an inverse Fourier transformation of a spectrum. The result is a single channel spectrum data block, which will be added to the file.

InverseFT ([<File>:BlockID] ...., {...});

Parameter	Value	Description
RSY		symmetry
	0	symmetric
	1	Antisymmetric
RXS	number	X-start frequency
RXE	number	X-end frequency
RWR	0	frequency limits used
	1	file limits used

7.7.13 KramersKronig

Performs a Kramers Kronig transformation.

This command performs a Kramers-Kronig transformation of a reflectance spectrum. The real and imaginary part of an absorbance-like spectrum will be calculated. The result is a single channel spectrum data block which will be added to the file.

KramersKronig ([<File>:BlockID] ...., {...});

Parameter	Value	Description
KKR		desired result
	0	refractive index (complex)
	1	absorbance
	2	dielectric function (complex)
	3	phase
KKS	number	X-start frequency
KKE	number	X-end frequency
KWR	0	use specified frequency limits
	1	use file limits

7.7.14 MakeCompatible

Makes spectra compatible.

This functions interpolates the selected spectrum to the frequency limits and point raster of a reference spectrum.

This functions modifies the selected spectrum and changes the data block type accordingly. The reference spectrum remains unchanged.

MakeCompatible ([<File1>:BlockID1], ([<File2>:BlockID2] ...., {...});

*<File1>*reference file.

<File2> file to be interpolated.

Parameter	Value	Description
CME		interpolation method
	2	interpolation
	3	reduce resolution

7.7.15 Merge

This function has not been implemented yet.

Merges spectra.

Merge ([<File>:BlockID] ...., {...});

7.7.16 Normalize

Normalizes a spectrum.

This functions modifies the selected spectrum.

Normalize ([<File>:BlockID] ...., {...});

Parameter	Value	Description
NME	1	min-max normalization
	2	vector normalization
	3	offset correction
NWR	0	use specified frequency limits
	1	use file limits
NFX	number	X-start frequency
NLX	number	X-end frequency

7.7.17 PostFTZerofill

Performs a post Zerofilling of a spectrum.

This functions modifies the selected spectrum.

PostFTZerofill ([<File>:BlockID] ...., {...});

Parameter	Value	Description
PZF	pos. number	Zerofilling Factor
PZS	number	X-start frequency
PZE	number	X-end frequency
PWR		frequency limits
	0	use specified frequency limits
	1	use file limits

7.7.18 RamanCorrection

Performs a Raman correction.

This functions modifies the selected spectrum.

RamanCorrection ([<File>:BlockID] ...., {...});

Parameter	Value	Description
QC0		background correction
	0	do not perform correction
	1	perform correction
QC1		scatter correction
	0	do not perform correction
	1	perform correction
QC2		restore original data
	0	do not perform correction
	1	perform correction
QC3	Text	path for white light source spectrum
QC4	Text	name of white light source spectrum
QC5	pos. number	reference temperature

7.7.19 Smooth

Smoothes a spectrum.

This functions modifies the selected spectrum.

Smooth ([<File>:BlockID] ...., {...});

Parameter	Value	Description
QSP	5, 9, 13, 17, 21, 25	number of smoothing points

7.7.20 StraightLine

Generates a straight line.

This functions modifies the selected spectrum.

StraightLine ([<File>:BlockID] ...., {...});

Parameter	Value	Description
GFX	number	X-start frequency
GLX	number	X-end frequency

7.7.21 Subtract

Subtracts one or more spectra from another spectrum.

The spectrum from which the others are subtracted is modified. The spectrum/ spectra which are subtracted stay unchanged.

Subtract ([<File A>:BlockIDA], ([<File B>:BlockIDB] ...., {...});

<*File A*>file to be subtracted from, this file is modified.

<*File B*> file(s) which are subtracted from <FileA>**.**

Parameter	Value	Description
SUB		subtraction mode
	1	interactive
	3	autosubtraction
	4	use whole range
SUN		number of spectra
SX1		X-start frequency
SX2		X-end frequency

7.8 OPUS Evaluation Functions

7.8.1 Integrate

Integrates a spectrum.

This function adds an integration report to the file.

Integrate ([<File>:BlockID] ...., {...});

Parameter	Value	Description
LPT	text	path for integration method
LFN	text	file name of the integration method
LRM		report mode

	0	overwrite old integration report
	1	merge integration reports
	2	append integration report

7.8.2 PeakPick

Creates a peak table.

This function adds a peak table data block to the file.

PeakPick ([<File>:BlockID] ...., {...});

Parameter	Value	Description
PSM		peak mode
	1	standard peak pick
	2	2. derivative
NSP	5, 9, 13, 17, 21, 25	number of points used for 2. derivative
WHR		frequency limits
	0	use specified frequency limits
	1	use file limits
LXP	number	start frequency
FXP	number	end frequency
PPM		peak definition
	1	autodetect (min or max)
	2	find maximum
	3	find minimum
PTR	pos. number	find peaks > value (absolute)
QP0		decimals
	YES	digits after decimal, user-defined
	NO	digits after decimal, not defined by user
QP3	pos. integer	digits after decimal
QP4		peak limits (%)
	YES	use peak limits
	NO	ignore peak limits
QP5	pos. integer	find peaks < value (%)
QP6		upper absolute peak limit
	YES	use upper absolute peak limit
	NO	ignore upper absolute peak limit
QP7	pos. integer	find peaks < value (absolute)

QP8		lower absolute peak limit
	YES	use lower absolute peak limit
	NO	ignore lower absolute peak limit

7.8.3 SignalToNoise

Calculates the Signal-to-Noise ratio.

This function adds parameters to the data parameter block of the selected spectrum.

SignalToNoise ([<File>:BlockID] ...., {...});

Parameter	Value	Description
NF1	number	start frequency
NF2	number	end frequency
SN1	number	S/N (RMS)
SN2	number	S/N (peak to peak)
SN3	number	maximum ordinate in S/N region
SN4	number	minimum ordinate in S/N region
SNF		flags

7.9 OPUS File Functions

7.9.1 ChangeDataBlockType

This function has not been implemented yet.

Change the data block type.

This functions does not modify the specified data block, only the block ID is changed.

ChangeDataBlockType ([<File>:BlockID] ...., {...});

7.9.2 CopyDataBlock

This function has not been implemented yet.

Copies a data block from one file to another.

This function adds the specified data block to the selected file in file list B.

CopyDataBlock ([<File A>:BlockID], ([<File B>] ...., {...});

<*file A*>source file.

*blockID*name of the data block to copy.

<*file B*>destination file.

7.9.3 DeleteDataBlock

Deletes the specified data block.

The specified block is removed from the file.

DeleteDataBlock ([<File>:BlockID] ...., {...});

7.9.4 Restore

Restores original File.

This function restores the original file and discards all changes made so far. All changes are lost if the results had not been saved before.

Restore([<File>:BlockID] ...., {});

The function does not require any parameters.

7.9.5 Save, SaveAs

Saves a spectrum file.

This function stores the eventually modified file to disk.

Save ([<File>:BlockID] ...., {...});

Save As ([<File>:BlockID] ...., {...});

Parameter	Value	Description
OEX		overwrite mode
	0	increment file name
	1	overwrite file
SAN		file name
DAP		target directory
COF		bit combination for save mode
	2	save all data blocks
	4	move file

	16	remove copies
	32	save as JCAMP.dx file
	64	save as x,y table
	128	replace original data
	256	save as Galactics GRAMS file
	512	unload file after saving it
	1024	save as Pirouette file
The following parameters will only be used when saving a file as an x,y table.		
DPA	pos. number	number of decimals, abscissa
DPO	pos. number	number of decimals, ordinate
SEP	character	separator
YON		Y-values
	1	Y-Values only
	0	X and Y-Values
ADP		data points
	1	use all data points
	0	do not use all data points

7.9.6 SendFile

Sends a file via e-mail.

This function does not modify the specified file.

SendFile ([<File>:BlockID] ...., {...});

Parameter	Value	Description
COF		data blocks
	0	send only specified block
	2	send all blocks

7.9.7 Unload

Removes a spectrum from Browser.

This function removes the specified file from the OPUS file list. The file is no longer accessible from the macro.

Unload ([<File>:BlockID] ...., {...});

The function does not require any parameters.

7.10 OPUS Measurement Functions

We strongly recommend to set the measurement parameters for a macro using an experiment file. Most of the parameter are linked and checked for consistency before starting an acquisition. Therefore, an inconsistent or wrong parameter set will most likely not be able to start an acquisition, and can be recognized easily. Only a few of the parameters listed below can be set without any problems either manually or by using variables.

7.10.1 Measurement Commands

The measurement commands always use the same parameters. You should only use the parameters listed here.

1) Measure Reference: ***MeasureReference ({...})***; acquires a background spectrum.
2) Measure Sample: <***File***> = ***MeasureSample ({...})***; acquires a sample spectrum.
3) Measure Repeated: <***File***> = ***MeasureRepeated ({...})***; acquires a set of sample spectra.
4) Measure Rapid TRS: <***File***> = ***MeasureRapidTRS ({...})***; performs a rapid scan acquisition.
5) Measure Step Scan Trans: <***File***> = ***MeasureStepScanTrans ({...});*** performs a Step Scan acquisition, using a transient recorder.

Parameter	Value	Description
SNM	text	sample name
SFM	text	sample preparation
CNM	text	operator
XPM	text	experiment file name
XPP	text	path for experiment file
RES	pos. number	resolution
NSS	pos. number	number of scans

7.10.2 SendCommand

Sends an optics command to the optics bench.

This function does not need an input spectrum.

SendFile (0, {...});

Parameter	Value	Description
UNI	text	text to be sent

7.10.3 SaveReference

Saves a reference spectrum from the AQP to disk.

This function creates a new file.

SaveReference (0, {...});

The function does not require any parameters.

7.10.4 LoadReference

Loads a reference spectrum from disk into the AQP.

This function does not modify the spectrum.

LoadReference ([<File>:ScRf], {...});

The function does not require any parameters.

7.11 OPUS Library Functions

7.11.1 LibrarySearchInfo

Searches for information in a spectrum library.

This function performs a query for information within a spectrum library. The query text must be supplied in a query file (extension .INL); use the OPUS-NT *Information Search* dialog to create and save a query file.

[<File1>:BlockID] = LibrarySearchInfo (0, {...});

Standard search.

*<File 1>*Contains the search result.

[<File1>:BlockID] = LibrarySearchInfo ([<File2>:BlockID], {...});

Query using an existsing search report.

*<File 1>*Contains the search result.

<File 2> Contains the search report**.**

Parameter	Value	Description	Remarks
SIH	NUMERIC	maximum number of hits	must be > 0
SIN	STRING	name of the information query file	file name including extension (.INL)
SIP	STRING	path of the query file	path without teminating "\"
LB1	STRING	list of library files to be searched	names of the library files. They must be stated including drive and path but without extension. Separate multiple file names with "@".

7.11.2 LibrarySearchPeak

Searches for peaks in a spectrum library.

This function performs a query for peaks within a spectrum library. The query R must be supplied in a query file (extension .PKL); use the OPUS-NT *Peak Search* dialog to create and save such a query file.

[<File1>:BlockID] = LibrarySearchPeak (0, {...});

Standard search.

*<File 1>*Contains the search result.

[<File1>:BlockID] = LibrarySearchPeak ([<File2>:BlockID], {...});

Query using a search report.

*<File 1>*Contains the search result.

<File 2> Contains the search report.

Parameter	Value	Description	Remarks
SPQ	NUMERIC	minimum Hit quality	Range between 1 and 1000 Will only be used in combination with the *Calculate Hit Quality* algorithm.
SPH	NUMERIC	maximum Hit number	must be > 0
SPA	NUMERIC	search algorithm	

	512	hit if one peak matches	
	1024	hit if all peaks match	
	2048	calculate hit quality	
	4096	count matching peaks	
PNP	STRING	name of the peak query file	file name including extension (.PKL)
PPP	STRING	path of the peak query file	path without teminating "\"
LB1	STRING	list of library files to be searched	names of the library files. They must be stated including drive and path but without extension. Separate multiple file names with "@".

7.11.3 LibrarySearchStructure

Searches for chemical structures in a spectrum library.

This function performs a query for chemical structures within a library file. The query must be supplied in a structure data block.

LibrarySendStructure([<File1>:BlockID], 0, {...});

Standard search.

*<File 1>*Contains the query structure.

LibrarySearchStructure ([<File1>:BlockID], [<File1>:BlockID], {...});

Query using an existing search report. The result will be appended to the file containing the structure block.

*<File 1>*Contains the query structure.

<File 2> Contains the search report**.**

Parameter	Value	Description	Remarks
STH	NUMERIC	maximum number of Hits	must be > 0
LAL	NUMERIC	search algorithm	
	8192	match exact	

	12288	match embedded	
LB1	STRING	list of library files to be searched	names of the library files. They must be stated including drive and path but without extension. Separate multiple file names with "@".

7.11.4 LibrarySearchSpectrum

Searches for spectra in a spectrum library.

This function performs a query for peaks within a spectrum library. The query spectrum must be absorbance-like.

LibrarySearchSpectrum ([<File1>:BlockID], 0, {...});

Standard search.

*<File 1>*The query spectrum.

LibrarySearchSpectrum ([<File1>:BlockID], [<File2>:BlockID], {...});

Query using a search report.

*<File 1>*The query spectrum.

<File 2> Contains the search report**.**

Parameter	Value	Description	Remarks
LSS	NUMERIC	sensitivity	Range between 1 and 20 Will only be used in combination with the *Standard* algorithm.
SSQ	NUMERIC	minimum Hit quality	Range between 1 and 1000
SSH	NUMERIC	maximum number of Hits	must be > 0
SS1	NUMERIC	search algorithm	When using spectrum correlation algorithms, the value will always be the sum of three options.
	1	standard	
	2	standard, use existing peak table	
	4	spectrum correlation	

	+16	no derivative	one of the three derivatization types must be added to the base value.
	+32	first derivative	
	+64	second derivative	
	+128	vector normalization	one of the two normalization types must be added to the base value.
	+256	min-max normalization	
LB1	STRING	list of library files to be searched	names of the library files. They must be stated including drive and path but without extension. Separate multiple file names with "@".

7.11.5 LibraryInitialize

Creates a new, empty library.

A method file (extension .MTD) and a text file (extension .TXD) is needed to create a library file.

LibraryInitialize ({...});

The function does not require any parameters.

Parameter	Value	Description	Remarks
LPT	STRING	path of the text definition file	path without teminating "\"
LBT	STRING	name of the text definition file	file name without extension
MTP	STRING	path of the method file	path without teminating "\"
LMT	STRING	name of the method file	file name without extension
LBP	STRING	directory of the new library file	path without teminating "\"
LBN	STRING	name of the library file	file name without extension
LID	STRING	library description	maximum 79 characters
LCP	STRING	copyright	maximum 79 characters

7.11.6 LibraryStore

Stores a new entry, the library description, and the definition of information saved in a library file.

LibraryStore (0, [<File>:BlockID], {...});

The function does not require an input file list.

Parameter	Value	Description	Remarks
LSM	NUMERIC	storage mode	
	1	new entry	
	3	replace entry	
	5	replace info	
	7	insert/replace structure	
LBP	STRING	directory of the library file	path without teminating "\"
LBN	STRING	name of the library file	file name without extension
LBS	NUMERIC	entry number	for all storage modes except "New Entry".

7.11.7 LibraryEdit

This function loads and deletes entries of a library. Furthermore, the description of the library as well as the description of the stored information can als be edited.

[<File>:BlockID] = LibraryEdit (0, {...});

Syntax to load a spectrum of a library entry.

LibraryEdit (0, {...});

Syntax for any other option

.

Parameter	Value	Description	Remarks
LMO	NUMERIC	edit mode	
	2	load entry	
	5	delete entry	
	13	change information set	

	14	change description	
LBS	NUMERIC	entry number	only required for the “Load Entry” and “Delete Entry” mode.
LBP	STRING	directory of the library file	path without teminating “\”
LBN	STRING	name of the library file	file name without extension
LID	STRING	new information definintion file or library description	only required for the “Chande Info Definition” and “Chande Description” mode. Contains the complete path and name of the new information definition file (extension .TXD) or the description, depending on the mode.

7.11.8 InfoInput

Allows information input.

This function adds an information block to the selected file. Depending on the mode, either the complete info block is replaced, only selected information of an existing info block is replaced, or a new file with an info block will be created.

InfoInput ([<File>:BlockID] ...., {...});

Syntax if a block should be replaced or extended.

[<File>:BlockID] = InfoInput ({...});

Syntax if a new file should be created.

Parameter	Value	Description	
IRM	STRING	information input mode	a
	O	the complete info block will be over-written.	
	R	the complete info block will be replaced.	

	N	generate new info file	
INP	STRING	path of the info definition file	Required for the modes “O” and “N”.
INM	STRING	name of the info definition file	Required for the modes “O” and “N”.
I01	STRING	information of line 1	b
I02	STRING	information of line 2	
...			
I99	STRING	information of line 99	
T01	STRING	description of line 1	
T02	STRING	description of line 2	
...			
T99	STRING	description of line 99	

a. If stated, make sure to consider the following points:

The parameter IRM is not allowed in the parameter list.

Null strings have to be assigned to the parameters INM and INP (e.g. INM = ’’)

The paramters Txx have to be specified consecutively, starting with T00. For example, in case of 4 lines, the parameters T00, T01, T02, T03, T04 must be stated.

The parameters Ixx responsible for the line content, like all other options, don’t need to be specified consecutively.

b. Specify the text to be entered in the info block using the parameters Ixx. xx represents the line numbers in the info block. You only have to state parameters for the lines in which you wish to enter text. The total number of lines is defined in the info definition file.

7.12 Miscellaneous OPUS Functions

7.12.1 ExternalProgram

Starts an external program.

This function launches an external program, forwards parameters and supplies the means of communication with the external program. DDE connections as well as Named Pipes are supported.

ExternalProgram ([<File>:BlockID] ...., {...});

Parameter	Value	Description
XPF		start as OPUS task
	0	OPUS starts the program, then breaks off all communictation with the external program
	1	program is not detached
XST		type of program start and connection type
	0	start the program; connection via a pipe
	1	don't start the program; connection via the server pipe
	2	start the program; open a DDE connection
	3	don't start the program; connection via the server pipe
XPR	Text	name of the program to be launched, including path
XPA	Text	parameteres to be exchanged
XWI		start 16bit program in its own VDM
	0	use common VDM
	1	extra VDM
XWS		window size at start
	0	normal
	1	maximized
	2	minimized
	3	hidden
XCW		wait for program termination
	0	only start program
	1	wait for result/end
XSB		start in background mode – not supported by Windows NT. Can be replaced by XWS
XEM		OS/2 spezific – no longer supported
XDM		OS/2 spezific – no longer supported
XVP		OS/2 spezific – no longer supported
XPM	<C/S>	OS/2 spezific – no longer supported
DDE		transaction type
	Bit 0 gelöscht	don't send command
	1	poke

	3	execute
	5	request
DDS	Text	DDE server name
DDT	Text	DDE topic
DDI	Text	DDE item
DDD	Text	text-coded binary data

7.12.2 ParameterEditor

Changes the sample parameters.

This function changes the following parameters:

- sample name
- sample form
- user name
- sample number

Note that the statement of all values is required when executing this function. In addition, the axes labels and scaling factors used for the axes can be entered.

ParameterEditor ([<File>:BlockID] ...., {...});

Parameter	Value	Description
CNM	Text	user name
SNM	Text	sample nname
SFM	Text	sample form
RSN	Zahl	sample number
XTX	Text	X-axis label
YTX	Text	Y-axis label
ZTX	Text	Z-axis label
XAF	Number	X-axis scaling factor
YAF	Number	Y-axis scaling factor
ZAF	Number	Z-axis scaling factor

7.12.3 Plot

Plots spectra.

This function does not change the spectrum.

Plot ([<File>:BlockID] ...., {...});;

Parameter	Value	Description
PDV		output device
	Printer	printer
	Clipboard	clipboard
SCP	Text	path of the template used for plotting
SCN	Text	name of the template used for plotting
PUN		devices; currently not evaluated
POP	Text	output path; currently not evaluated
POF	Text	output file; currently not evaluated
PDH		window handle; currently not evaluated
PL2	Number	number of peaks to be labeled
PPA	Text	Codes several parameters in a string that are used for different frames

PPA starts with FRM=n and defines how many frame parameters follow. For each frame the following parameters (separated by commas) are necessary:

Parameter	Value	Description
NPL	Number	number of spectra in the current frame
XSP	Number	X start frequency
XEP	Number	X end frequency
YMN	Number	lowest value of the Y axis
YMX	Number	highest value of the Y axis
ASE	YES/NO	AutoScale the spectrum frame
CWN	YES/NO	use compressed wave numbers
COL	Numbers	colors of each curve, separated by commas

7.12.4 VBScript

Starts a VisualBasic script

This function loads and then runs a VisualBasic script. Parameters and data blocks can be forwarded to the script

VBScript ([<File>:BlockID] ...., {...});

Parameter	Value	Description
VBS	Text	name of the script, including path
VBP	Text	parameters to be forwarded to the script
VBW		wait for termination
	0	immediate return after starting the script
	1	wait for result/end
VBH		start in background mode
	0	start in foreground – script will be displayed
	1	start in background – script will not be displayed

8 Client/Server Reference

The Client/Server Interpreter is the module of OPUS responsible for processing commands received through the Pipe-, DDE- or Scripting interface. Therefore, the list of commands is the same for all three interfaces.

The following chapters mainly address users who intend to write their own programs and link them to OPUS or OPUS macros. This is achieved with the OPUS command *External Program*, which was described earlier. In the following we expect the user to be familiar with this command and its options.

A part of these commands was already available under OPUS OS/2 in form of the Client/Server function. Hence, in the following the commands are divided in old and new ones.

8.1 Overview of Available Functions

Currently, you can use a client program to:

- read data from OPUS spectrum files and 3D files; you can either read the whole frequency region or select a part of interest from the data.
- write data to OPUS spectrum files and 3D files; you can either write the whole frequency region or select a part of interest from the data.
- load and unload OPUS files.
- read file information from the Client/Server file list.
- read OPUS parameters from an OPUS file.
- save OPUS parameters to an OPUS file.
- read data from report blocks.
- start OPUS macros.
- exchange parameters with an OPUS macro.

In addition, all functions of the command line, i.e. all OPUS processing functions are supported, according to the syntax described earlier.

8.2 Commands and Command Syntax

In the following you find a list containing all Client/Server commands. The description of all commands is structured in the same manner:

Syntax:

The name of the command and the syntax that has to be applied. Mandatory exchange parameters are indicated with „< >“, optional parameters are enclosed in square brackets „[ ]“.

Description:

A description of the action performed by the command.

Return Value:

A list of the possible return values.

Return Value 2:

Return Value 3:

Some commands return additional text after confirming the execution with *OK*; in this case they must be read.

Errors:

A list of possible error messages.

Comments:

Notes and further comments about the command.

8.3 Old C/S Commands

These commands have been available already in OPUS-OS/2.

8.3.1 Overview

The following commands are still used by OPUS-NT:

TIMEOUT	sets the maximum wait time
CLOSE_PIPE	closes pipe
OVERWRITE	activates overwrite mode
PRESERVE	deactivates overwrite mode
COUNT_ENTRIES	counts entries of the file input list
READ_FROM_ENTRY	sets the entry number and data block
WRITE_TO_ENTRY	sets the entry number and data block for writing
READ_FROM_FILE	selects file for reading
WRITE_TO_FILE	selects file for writing
READ_FROM_BLOCK	specifies the data block for reading
WRITE_TO_BLOCK	specifies the data block for writing
ASCII	sets data point mode to text
BINARY	sets data point mode to binary
DATA_VALUES	sets data point mode to frequencies
DATA_POINTS	sets data point mode to data points
READ_HEADER	reads spectrum header
READ_DATA	reads spectral data
WRITE_HEADER	writes spectrum header

WRITE_DATA	writes spectral data
COPY_DATA	copies spectrum block
LOAD_FILE	loads a file
UNLOAD_FILE	unloads a file
START_MACRO	runs a macro
FILE_PARAMETERS	sets parameter mode to spectrum parameters
OPUS_PARAMETERS	sets parameter mode to OPUS parameters
READ_PARAMETER	reads parameters

8.3.2 CLOSE_PIPE

Syntax:

"CLOSE_PIPE"

Description:

Closes the pipe connection.

Return Value:

"OK"

Comment:

Although it is not strictly required, this command should be send if no further communication with OPUS is necessary. The corresponding program pipe will be closed by OPUS and the resources returned.

8.3.3 COUNT_ENTRIES

Syntax:

"COUNT_ENTRIES"

Description:

Returns the number of data blocks that have been selected in the *Select File* dialog of the *External Program* function.

Return Value:

"OK"

Return Value 2:

<Number of data blocks>

Comment:

This command ensures that all files or data blocks selected in the *Select File* dialog of the *External Program* function can be accessed.

8.3.4 READ_FROM_ENTRY

Syntax:

“READ_FROM_ENTRY <Number>”

Description:

This command specifies the data block accessed by the READ_DATA command.

Return Value:

“OK” or error message.

Error:

“Syntax: READ_FROM_ENTRY <Number>”

“Entry number out of range”

Return Value 2:

<Complete path of the OPUS file>

<File number>

<Data block name>

Comment:

The argument to this command is the number of an entry in the Client/Server file list (between 1 and the number returned from the COUNT_ENTRIES command), from which the client program intends to read. If no error occurs, the complete file name (including drive and path), as well as the data block name of the selected file in text format will be returned as the second return value. The format of the data block name is identical to the one used in the history function.

The file name returned by the command is hyphenated and followed by the number of the copy (clonecount) for further use with the command line.

8.3.5 WRITE_TO_ENTRY

Syntax:

“WRITE_TO_ENTRY <Number>”

Description:

This command specifies the data block accessed by the WRITE_DATA command.

Return Value:

"OK" or error message.

Error:

"Syntax: WRITE_TO_ENTRY <Number>"

"Entry number out of range"

Return Value 2:

<Complete path of the OPUS file>

<File number>

<Data block name>

Comment:

The argument to this command is the number of an entry in the Client/Server file list (between 1 and the number returned from the COUNT_ENTRIES command). If no error occurs, the complete file name (including drive and path), as well as the name of the data block in text format will be returned as the second return value. The name of the data block is returned in the same format used in the history function.

The file name returned by the command is hyphenated and followed by the number of the copy (clonecount) for further use with the command line.

8.3.6 READ_FROM_FILE

Syntax:

"READ_FROM_FILE <Filename> or <File number>"

Description:

Specifies the OPUS file from which the client program intends to read. The argument to this command is the file name which can be specified with or without hyphens. Optionally, the clonecount can be stated. If the file was already loaded in OPUS using this name (including the correct clonecount), this copy will be used. Otherwise, the file will automatically be loaded. For reasons of compatibility to OPUS-OS/2 the file can still be accessed via an internal file number, but this number is no longer limited to the region between 1 to 699.

Return Value:

"OK" or error message.

Error:

“Syntax: READ_FROM_FILE <File name> or <File number>”

“File not Found”

Return Value 2:

<Complete path of the OPUS file>

<File number>

Comment:

Specifies the OPUS file from which the client program intends to read. This command is only able to select a file; the READ_FROM_BLOCK command must subsequently be used to specify the data block in the file, from which to read.

The error message „File not Found“ can have multiple causes. In general, it indicates an error while accessing the file.

The file name returned by the command is hyphenated for further use in the command line and is followed by the number of the copy (clonecount).

8.3.7 WRITE_TO_FILE

Syntax:

“WRITE_TO_FILE <File name> or <File number>”

Description:

Specifies the OPUS file to which the client program intends to write. The argument to this command is the file name which can be specified with or without hyphens. Optionally, the clonecount can be stated. If the file was already loaded in OPUS using this name (including the correct clonecount), this copy will be used. Otherwise, the file will automatically be loaded. For reasons of compatibility to OPUS-OS/2 the file can still be accessed via an internal file number, but this number is no longer limited to the region between 1 to 699.

Return Value:

“OK” or error message.

Error:

“Syntax: WRITE_TO_FILE <File name> or <File number>”

“File not Found”

Return Value 2:

<Complete path of the OPUS file>

<File number>

Comment:

Specifies the OPUS file to which the client program intends to write. This command is only able to select a file; the WRITE_TO_BLOCK command must subsequently be used to specify the data block in the file to which to write.

The error message „File not Found“ can have multiple causes. In general, it indicates an error while accessing the file.

The file name returned by the command is hyphenated for further use in the command line and is followed by the number of the copy (Clonecount).

8.3.8 READ_FROM_BLOCK

Syntax:

“READ_FROM_BLOCK <Block name>”

Description:

Specifies the data block from which the client program intends to read. The command always refers to the file that was last specified with the READ_FROM_ENTRY or the READ_FROM_FILE command.

Return Value:

“OK” or error message.

Error:

“Syntax: READ_FROM_BLOCK <Block name>”

“No Filename or Filenumber defined”

“Unknown blocktype”

“Block not found”

Comment:

The argument to the command is the block type which is also used in reports i.e. “AB” for an absorption spectrum, “TR/Multiple” for a transmission block of a 3D file. The command will only be accepted if it was preceded by either the READ_FROM_ENTRY or the READ_FROM_FILE command.

8.3.9 WRITE_TO_BLOCK

Syntax:

“WRITE_TO_BLOCK <Block name>”

Description:

Specifies the data block from which the client program intends to write. The command always refers to the file that was last specified with the WRITE_TO_ENTRY or the WRITE_TO_FILE command.

Return Value:

“OK” or error message.

Error:

“Syntax: WRITE_TO_BLOCK <Block name>”

“No Filename or Filenumber defined”

“Unknown blocktype”

“Block not found”

Comment:

The argument to the command is the block type which is also used in reports, i.e. “AB” for an absorption spectrum, “TR/Multiple” for a transmission block of a 3D file. The command will only be accepted if it was preceded by either the WRITE_TO_ENTRY or the WRITE_TO_FILE command.

8.3.10 ASCII

Syntax:

“ASCII”

Description:

Sets the transfer mode used to transfer data points to ASCII.

Return Value:

“OK”

Comment:

If this mode is chosen (default mode) all data points will be transferred as ASCII text. Each data point is followed by an End of Line sequence.

8.3.11 BINARY

Syntax:

"BINARY"

Description:

Sets the transfer mode used to transfer data points to BINARY.

Return Value:

"OK"

Comment:

If this mode is chosen, all data points will be transferred as 4 byte IEEE floating-point number (REAL*4 in FORTRAN, FLOAT in C). In this mode, the data points will not be terminated. Therefore, the number of bytes transferred is N*4, N being the total number of transferred data points. This mode is faster than the ASCII mode.

8.3.12 DATA_VALUES

Syntax:

"DATA_VALUES"

Description:

The parameters of the READ_HEADER, READ_DATA and COPY_DATA will be interpreted as frequency values.

Return Value:

"OK"

8.3.13 DATA_POINTS

Syntax:

"DATA_POINTS"

Description:

The parameters of the READ_HEADER, READ_DATA and COPY_DATA will be interpreted as data points.

Return Value:

"OK"

Comment:

The data point numbering starts with ,,1". Floating-point numbers are always rounded to the next lower integer (e.g. 14.965 will be rounded to 14).

8.3.14 READ_HEADER

Syntax:

"READ_HEADER [<X1>[-<X2>] [<Z1>[-<Z2>]]"

Description:

Reads the header of a spectrum block and returns the frequency range of the spectrum. Several options are available.

Return Value:

"OK" or error message.

Error:

"No Filename or Filenumber defined"

"No Blocktype defined"

"Error Reading File"

"Not implemented"

Return Value 2:

In case of regular spectrum blocks:

<Number of data points (NX = XL - XF + l) in region X>

<Frequency of the first data point in region X>

<Frequency of the last data point in region X>

Return Value 2:

In case of 3D spectrum blocks:

<Number of data points (NX = XL - XF + I) in region X>

<Frequency/number of the first data point in region X>

<Frequency/number of the last data point in region X>

<Number of spectra (NZ = ZL - ZF + 1) in region Z>

<Value (e.g. time) of the first spectrum in region Z>

<Value (e.g. time) of the last spectrum in region Z>

Note: The output will always be returned as ASCII text, separated by an End of Line sequence, regardless of the selected data transfer mode.

Comment:

Up to four parameters can be forwarded as command arguments.

<X1>, <X2> define the frequency region of the spectrum block. If <X2> is not explicitly stated, only one data point in the vicinity of <X1> will be returned. If no parameters are specified or if <X1> was set to „*", all data stored in the spectrum block will be returned.

<Z1>, <Z2> define the region of the Z axis for which data will be returned (only for 3D files). If <Z2> is not specified, only data in the vicinity of <Z1> will be returned. If no parameters are specified or if <Z1> was set to „*", all data stored in the spectrum block will be returned. In the case of regular spectrum blocks, the parameters <Z1> and <Z2> will be ignored and do not cause an error message in case they have been stated.

All four parameters can either be entered as integer or as floating-point number and will be interpreted either as frequencies or as data points, depending on the settings (see the DATA_VALUES and DATA_POINTS commands).

8.3.15 READ_DATA

Syntax:

"READ_DATA [<X1>[<X2>]] [<Z1>[-Z2]]"

Description:

Reads the header and data points of a spectrum block within the limits indicated. The parameters of the command are similar to the parameters of the READ_HEADER command.

Return Value:

"OK" or error message.

Error:

"No Filename or Filenumber defined"

"No Blocktype defined"

"Error Reading File"

"Not implemented"

Return Value 2:

In case of regular spectrum blocks:

<Number of data points (NX = XL - XF + l) in region X>

<Frequency of the first data point in region X>

<Frequency of the last data point in region X>

<Scaling factor for Y values>

<Y(XF)>, <Y(XF + 1)>, <Y(XF + 2)> ...<Y(XL)>

“OK” or “Error Reading File”

Return Value 2:

In case of 3D spectrum blocks:

<Number of data points (NX = XL - XF + 1) in region X>

<Frequency/number of the first data point (XF) in region X>

<Frequency/number of the last data point (XL) in region X>

<Number of spectra (NZ = ZL - ZF + 1) in region Z>

<Value (e.g. time) of the first spectrum in region Z>

<Value (e.g. time) of the last spectrum in region Z>

<Scaling factor for Y values> for Z = ZF

<Y(XF)>, <Y(XF + 1)>, ... <Y(XL)> for Z = ZF

<Scaling factor for Y values> for Z = ZF+1

<Y(XF)>, <Y(XF + 1)>, ... <Y(XL)> for Z = ZF + 1

<Scaling factor for Y values> for Z = ZF + 2

<Y(XF)>, <Y(XF + 1)>, ... <Y(XL)> for Z = ZF + 2

...

<Scaling factor for Y values> for Z = ZL

<Y(XF)>, <Y(XF + 1)>, ... <Y(XL)> for Z = ZL

“OK1” or “Error Reading File”

Comment:

The header values will always be returned as ASCII text, separated by an End of Line sequence, regardless of the selected data transfer mode. The data points will be returned either as ASCII text, separated by an End of Line sequence, or as floating-point numbers without any separator, depending on the selected data transfer mode. Either ,,OK” or the error message “Error Reading File” will be appended after the data points.

8.3.16 WRITE_HEADER

Syntax:

“WRITE_HEADER”

Description:

Writes a (new) header for a data block. After the command, the following parameters must be send as ASCII text:

Return Value:

In case of regular spectrum blocks:

<Number of data points (NX = XL - XF + l) in region X>

<Frequency/number of the first data point in region X>

<Frequency/number of the last data point in region X>

In case of 3D spectrum blocks:

<Number of data points (NX = XL - XF + I) in region X>

<Frequency/number of the first data point in region X>

<Frequency/number of the last data point in region X>

<Number of spectra (NZ = ZL - ZF + 1) in region Z>

<Value (e.g. time) of the first spectrum in region Z>

<Value (e.g. time) of the last spectrum in region Z>

“OK” or error message.

Error:

“No Filename or Filenumber defined”

“No Blocktype defined”

“Not implemented”

Comment:

This command serves to edit existing data block headers. Especially, pay attention to the number of data points (especially in Z direction): the number of data points specified must match the actual number of data points stored in the data block. Otherwise, a shift of the data will result.

8.3.17 WRITE_DATA

Syntax:

“WRITE_DATA”

Description:

Writes the header and data points into a data block. After the command, the following parameters must be send:

In case of regular spectrum blocks:

<Number of data points (NX = XL - XF + l) in region X>

<Frequency/number of the first data point (XF) in region X>

<Frequency/number of the last data point (XL) in region X>

<Scaling factor for Y-Values> <Y(XF)>, <Y(XF + 1)>, <Y(XF + 2)> ...<Y(XL)>

In case of 3D spectrum blocks:

<Number of data points (NX = XL - XF + I) in region X>

<Frequency/number of the first data point (XF) in region X>

<Frequency/number of the last data point (XL) in region X>

<Number of spectra (NZ = ZL - ZF + 1) in region Z>

<Value (e.g. time) of the first spectrum in region Z>

<Value (e.g. time) of the last spectrum in region Z>

<Scaling factor for Y values> for Z = ZF <Y(XF)>, <Y(XF + I)>, <Y(XI,)> for Z = ZF

<Scaling factor for Y values> for Z = ZF + 1

<Y(XF)>, <Y(XF + 1)>, ... <Y(XL)> for Z = ZF + 1

<Scaling factor for Y values> for Z = ZF + 2

<Y(XF)>, <Y(XF + 1)>, ... <Y(XL)> for Z = ZF + 2

...

<Scaling factor for Y values> for Z = ZL

<Y(XF)>, <Y(XF + I)>, ... <Y(XL)> for Z = ZL

Return Value:

"OK" or error message.

Error:

"No Filename or Filenumber defined"

"No Blocktype defined"

"Not implemented"

"Error Accessing Data"

Return Value 2:

After the header and all data points have been read by OPUS, either "OK" or an error message will be returned.

Comment:

The header values must always be sent as ASCII text, separated by an End of Line sequence, regardless of the selected data transfer mode. The data points must be returned either as ASCII text, separated by an End of Line sequence, or as floating-point numbers without any separator, depending on the selected data transfer mode.

8.3.18 COPY_DATA

Syntax:

"COPY_DATA [<X1>[-<X2>]] [<Z1>[-<Z2>]]"

Description:

Copies data points from a data block specified by one of the commands READ_FROM_ENTRY or READ_FROM_FILE and READ_FROM_BLOCK to a data block specified by either the WRITE_TO_ENTRY or WRITE_TO_FILE and WRITE_TO_BLOCK command (for parameters see READ_HEADER).

Return Value:

After receiving the command:

"OK" or error message.

Return Value 2:

After processing the command:

"OK" or error message.

Error:

“No Filename or Filenumber defined”

“No Blocktype defined”

“Not implemented”

“Error Reading File”

Comment:

The copy process take place within OPUS. Therefore, no data points are transferred via a pipe.

8.3.19 LOAD_FILE

Syntax:

“LOAD_FILE <File name>”

Description:

Loads the indicated file into OPUS.

Return Value:

“OK” or error message.

Error:

“Syntax: LOAD_FILE <File name>”

“Error reading file”

Return Value 2:

<Path and name of the file>

<File number>

Comment:

OPUS loads the file even if it has already been loaded before. In this case another copy (clone) is generated.

The file name returned by the command is hyphenated for further use in the command line and is followed by the number of the copy (Clonecount).

8.3.20 UNLOAD_FILE

Syntax:

“UNLOAD_FILE <File name> or <File number>”

Description:

Unloads a file from OPUS selection line. The argument to this command is the file name (including clonecount).

Return Value:

“OK” or error message.

Error:

“Syntax: UNLOAD_FILE <File name> or <File number>”

“File not loaded”

Return Value 2:

<Path and name of the file>

<File number>

Comment:

OPUS unloads the selected file. The complete path and file name, as well as the entry number will be returned. If the path of the file is not specified, OPUS searches the “Data Path” directory for the file.

For reasons of compatibility to OPUS-OS/2 the file can still be accessed via an internal file number, but this number is no longer limited to the region between 1 to 699.

The file name returned by the command is hyphenated for further use in the command line and is followed by the number of the copy (clonecount).

8.3.21 START_MACRO

Syntax:

“START_MACRO <Macro file name>[<Number of input parameters>]”

Description:

Starts an OPUS macro. Input parameters can be forwarded to the macro. If parameters are exchanged, the total number of parameters must be defined as the second parameter. If this number is omitted, then it will be set to 0; in this case, no parameters are read. If the number of input parameters is larger than 0,

the input parameters <input parameter 1>, <input parameter 2>, ..., <input parameter N> have to be sent by the client program. In addition, the macro can return parameter values to the client program.

Return Value:

Immediately after the command execution (i.e. directly after the starting the macro):

“OK” or error message.

Return Value 2:

After macro execution:

“OK” or error message.

Return Value 3:

only if the macro returned parameters:

<Number of return value parameters>

<Return value parameter 1>

< Return value parameter 2>

...

<Return value parameter N>

Return Value 4:

only if the macro returned parameters:

“OK” or error message.

Error:

“Syntax: START_MACRO <Macro File> <#Parameter>”

“Error in Opus Command Line Execution - ID: %d”

Comment regarding the command:

When executing the command the following order has to be maintained:

- Send command including the macro name and the number of input parameters (optional).
- Read return value: “OK” or error message.
- Send input parameter.
- Read return value: “OK” or error message.
- Read return value parameter.

The individual input parameters must be separated by End of Line sequences.

Macro parameters can also directly follow a command; in this case the second "OK" or error message will not be send.

Comment regarding the macros:

A structure similar to sub macro calls is used to control client programs. Input parameters will be transferred from the client program to the macro using a dialog box, that must be located in the first line of the respective macro. Return values are returned via another dialog box located in the last line of the macro. As in the case of a sub macro call, both dialog boxes will not be displayed. If OPUS cannot find a dialog box in the first macro line, the macro will be started without exchanging parameters, even if they have been sent to OPUS.

In the first dialog box, the input parameters will be assigned from top to bottom; only variables of type FILE, TEXT FOR EDIT, NUMERIC, TEXT FOR OUTPUT or CHECK BOX are allowed. Empty lines and variables of type BUTTON and COMBOBOX will be ignored. If the number of input paramteres exchanged is not equal to the number of variables in the dialog box, OPUS terminates the assignment either after all input parameters have been read or if all macro variables have been assigned. ASCII input paramters will automatically be converted into the format of the macro variable. Accordingly, the return values will be transformed by the last dialog box in the macro from top to bottom into ASCII text, and, delimited by an End of Line character, returned to the client program. Here also, empty lines and variables of type BUTTON and COMBOBOX will be ignored. If no dialog box can be found in the last line of the macro (or if the dialog box is empty), OPUS returns "0" as number of return value parameters immediately after starting the macro. Communication will be resumed without waiting for the macro to terminate.

8.3.22 FILE_PARAMETERS

Syntax:

"FILE_PARAMETERS"

Description:

After this command, the READ_PARAMETER command reads a parameter from the data block of a file specified by the commands READ_FROM_ENTRY, READ_FROM_FILE or READ_FROM_BLOCK.

Return Value:

"OK"

Comment:

This is the default setting for the READ_PARAMETER command.

8.3.23 OPUS_PARAMETERS

Syntax:

"OPUS_PARAMETERS"

Description:

After this command, READ_PARAMETER reads a parameter from the OPUS default parameter set.

Return Value:

"OK"

8.3.24 READ_PARAMETER

Syntax:

"READ_PARAMETER <Parameter name>"

Description:

Reads a parameter either from a specified data block of an OPUS file or from the standard OPUS parameter set.

Return Value:

"OK" or error message.

Error:

"Syntax: READ_PARAMETER <parameter name>"

"No Filename or Filenumber defined"

"No Blocktype defined"

"Parameter not found"

"Invalid Parameter Name"

Return Value 2:

<Parameter value>

Comment:

The parameter name forwarded as argument to the command consists of a three-character abbreviation. After the confirmation by OPUS, the parameter value will be transferred as ASCII text.

8.3.25 WRITE_PARAMETER

Syntax:

“WRITE_PARAMETER <Parameter name> <Parameter value>”

Description:

The WRITE_PARAMETER command writes a parameter or changes an existing one in the OPUS file specified by either READ_FROM_ENTRY or READ_FROM_FILE and READ_FROM_BLOCK.

Return Value:

“OK” or error message.

Error:

“Syntax: WRITE_PARAMETER <Parameter name> <Parameter value>”

“No Filename or Filenumber defined”

“No Blocktype defined”

“Parameter not found”

“Invalid Parameter Name”

Comment:

The parameter name forwarded as argument to the command consists of a three-character abbreviation.

The parameter value will be forwarded as ASCII text file, i.e. numerical values have to be converted to ASCII strings.

8.3.26 RUN_MACRO

Syntax:

Equivalent to START_MACRO

Description:

The RUN_MACRO command starts a macro. Contrary to START_MACRO, the control is returned immediately after the macro was started. The RUN_MACRO command does not wait for the macro to terminate and also does not return any results.

Return Value:

After the command:

“OK” or error message.

Return Value 2:

After transferring the input parameter:

“OK” or error message.

Return Value 3:

<MacroID>: a macro identification number unique for each macro session.

Error:

Similar to START_MACRO

Comment:

See also START_MACRO.

To access the results of the macro started, the MACRO_RESULTS commands is used.

The returned <MacroID> is used as parameter for the MACRO_RESULTS and the KILL_MACRO commands.

8.3.27 MACRO_RESULTS

Syntax:

“MACRO_RESULTS <MacroID>”

Description:

The MACRO_RESULTS command retrieves the result parameters of a macro session that was started with the ID <MacroID>, using the RUN_MACRO command.

Return Value:

“OK” or error message.

Return Value 2:

0 or 1 to indicate whether the macro has already finished or is still running.

Return Value 3:

Containing the results, if the macro was terminated. For a format description see START_MACRO.

Error:

“Syntax: MACRO_RESULTS <MacroID>”

“Invalid Macro ID”

Comment:

In combination with the RUN_MACRO command, this command allows client programs to run different tasks while the macro is still running. Use this command to frequently check, whether the macro has finished and to obtain the return parameters.

8.3.28 KILL_MACRO

Syntax:

“KILL_MACRO <MacroID>”

Description:

KILL_MACRO terminates a macro session started by RUN_MACRO with the specified macro ID.

Return Value:

“OK” or error message.

Error:

“Syntax: KILL_MACRO <MacroID>”

“Invalid Macro ID”

Comment:

In combination with the RUN_MACRO command this command allows client programs to run different tasks while the macro is still running. Under certain conditions a client program can use this command to stop a macro that is still running. This corresponds to the *Abort Task* command of the OPUS task bar.

8.4 Obsolete Commands

The following commands are only supported out of compatibility reasons to OPUS-OS/2. Due to the different concept of OPUS-NT, they are no longer of any practical importance.

8.4.1 OVERWRITE

Syntax:

“OVERWRITE”

Description:

Allows the subsequent commands to overwrite files and data blocks.

Return Value:

“OK”

Comment:

Subsequent to this command, the following commands are allowed to overwrite files and data blocks:

WRITE_TO_ENTRY

WRITE_TO_FILE

WRITE_TO_BLOCK

8.4.2 PRESERVE

Syntax:

“PRESERVE”

Description:

Prevents files and data blocks from being replaced.

Return Value:

“OK” or error message.

Error:

“Set OVERWRITE mode to replace blocks”

Comment:

Subsequent to this command, the following commands cannot replace existing files and data blocks:

WRITE_TO_ENTRY

WRITE_TO_FILE

WRITE_TO BLOCK

If an existing data block was specified in a WRITE_TO_BLOCK command, OPUS returned the message “Set OVERWRITE mode to replace blocks”.

In case of a WRITE_TO_ENTRY or WRITE_TO_FILE command, the file name extension was incremented until the first non-existing file was obtained.

Example:

Assume the files TEST.2 and TEST.3 already exist in the current OPUS\DATA directory. The “WRITE_TO_FILE TEST.1” command is sent twice. The first time the command is executed and generates the file TEST.1. The second time, the file name extension is incremented until the first non-existing file name is obtained (TEST.1), because it is not allowed to replace the now existing file TEST.1.

8.4.3 TIMEOUT

Syntax:

“TIMEOUT <Delay>”

Description:

Sets a delay time (in seconds) for the pipe, which may not be replaced during read and write processes.

Return Value:

“OK” or error message.

Error:

“Invalid time limit”

“Syntax: TIMEOUT<Seconds>”

Comment:

The delay is an integer between l and 1000. Without this command the default value of 10 seconds will be used.

8.5 New Commands

The first view commands of this section serve to further specify the binary transfer mode. They mainly concern the data exchange with scripts. Because scripts allow no direct memory access, the data must be enclosed in a variable field to allow binary data exchange. Hence, the single elements are assigned a certain type: BYTE_MODE, INT_MODE, FLOAT_MODE, and DOUBLE_MODE allow to define, whether the binary OPUS data will be contained in a BYTE, INTEGER, FLOAT, or DOUBLE field in a script.

In case of a pipe, the respective memory region can be transferred directly, which then will be interpreted on the receiving side.

No binary return values are allowed when using DDE connections; these are available in the HEXSTRING_MODE.

8.5.1 BYTE_MODE

Syntax:

“BYTE_MODE”

Description:

Sets the binary transfer mode to single bytes.

Return Value:

“OK”

8.5.2 INT_MODE

Syntax:

“INT_MODE”

Description:

Sets the binary transfer mode to integer.

Return Value:

“OK”

8.5.3 FLOAT_MODE

Syntax:

“FLOAT_MODE”

Description:

Sets the binary transfer mode to floating-point numbers.

Return Value:

“OK”

8.5.4 DOUBLE_MODE

Syntax:

“DOUBLE_MODE”

Description:

Sets the binary transfer mode to double-precision.

Return Value:

“OK”

8.5.5 HEXSTRING_MODE

Syntax:

“HEXSTRING_MODE”

Description:

Sets the binary transfer mode to text.

Return Value:

“OK” or error message.

Comment:

DDE connection default settings for binary mode.

The data is converted to individual strings of numbers, depending on the mode chosen (BYTE_MODE, INT_MODE, FLOAT_MODE and DOUBLE_MODE) and will be transmitted as text.

8.5.6 FLOATCONV_MODE

Syntax:

“FLOATCONV_MODE ON|OFF”

Description:

Switches the conversion of floating-point numbers on and off, when using binary transfer mode.

Return Value:

“OK” or “ON|OFF”

Comment:

When using a pipe for binary data transfer under OS/2, a scaling factor was transferred prior to the actual data. This factor was also transferred binary, but compared to the data transfer at double-precision (8 instead of 4 bytes). In OPUS-NT, this factor is found in the first element of the returned field.

If FLOATCONV_MODE is not selected for the binary data transfer to a script, the first 8 bytes of data (the double-precision scaling factor) will be misinterpreted as two single-precision floating-point numbers.

If neither „ON“ nor „OFF“ is forwarded as parameter the return value text provides the current settings.

8.5.7 GET_DISPLAY

Syntax:

“GET_DISPLAY”

Description:

Provides an identification number of the currently active display window.

Return Value:

“OK”

Return Value 2:

<WindowID>

Comment:

The number returned can be used as parameter for the SET_WINDOW, CLOSE_WINDOW, and POSITION_WINDOW commands.

8.5.8 SET_WINDOW

Syntax:

“SET_WINDOW <WindowID>”

Description:

The window specified by the identification number will be promoted to be the active display window for the current C/S session.

Return Value:

“OK” or error message.

Error:

"Syntax: SET_WINDOW <Window>"

Comment:

If new files are loaded or generated by another OPUS function, they will be displayed in the currently active window. The function is used to define this window.

8.5.9 NEW_WINDOW

Syntax:

"NEW_WINDOW <Window type>"

Description:

Creates a new window of the type specified.

Return Value:

"OK" or error message.

Error:

"Syntax: NEW_WINDOW <Window type>"

"Error creating View"

Comment:

The window type defines, that for example a new report window will be generated.

8.5.10 CLOSE_WINDOW

Syntax:

"CLOSE_WINDOW <WindowID>"

Description:

Closes the window specified by the <WindowID>.

Return Value:

"OK" or error message.

Error:

"Syntax: CLOSE_WINDOW <Window>"

Comment:

The parameter <WindowID> necessary to address the display window can result from either NEW_WINDOW or from GET_DISPLAY.

8.5.11 POSITION_WINDOW

Syntax:

“POSITION_WINDOW <WindowID> <x> <y> <cx> <cy>”

Description:

Positions the display window specified by <WindowID> at the coordinates <x>, <y> and re-sizes it to <cx>, <cy>.

Return Value:

“OK” or error message.

Error:

“Syntax: POSITION_WINDOW <Window> <x> <y> <cx> <cy>\n”

Comment:

The parameter <WindowID>, necessary to address the display window, can result from either NEW_WINDOW or from GET_DISPLAY.

8.5.12 GET_LANGUAGE

Syntax:

“GET_LANGUAGE”

Description:

Retrieves the current language settings of OPUS-NT. The language is set using the command line argument /LANGUAGE when starting OPUS.

Return Value:

“OK”

Return Value 2:

<Language>

Comment:

The name of the language will be returned as text.

8.5.13 GET_OPUSPATH

Syntax:

"GET_OPUSPATH"

Description:

Retrieves the path of the currently running OPUS program.

Return Value:

"OK"

Return Value 2:

<Path>

Comment:

The path can be checked in the *User Settings* dialog box of the *Setup* OPUS pull-down menu.

8.5.14 GET_BASEPATH

Syntax:

"GET_BASEPATH"

Description:

Retrieves the default path of the currently logged in user.

Return Value:

"OK"

Return Value 2:

<Path>

Comment:

The path is set in the *User Settings* dialog box of the *Setup* OPUS pull-down menu.

8.5.15 GET_DATAPATH

Syntax:

"GET_DATAPATH"

Description:

Retrieves the data path of the currently logged in user.

Return Value:

“OK”

Return Value 2:

<Path>

Comment:

The path is set in the *User Settings* dialog box of the *Setup* OPUS pull-down menu.

8.5.16 GET_WORKPATH

Syntax:

“GET_WORKPATH”

Description:

Retrieves the path for work files of the currently logged in user.

Return Value:

“OK”

Return Value 2:

<Path>

Comment:

The path is set in the *User Settings* dialog box of the *Setup* OPUS pull-down menu.

8.5.17 GET_USERNAME

Syntax:

“GET_USERNAME”

Description:

Retrieves the name of the currently logged in user.

Return Value:

“OK”

Return Value 2:

<Name>

Comment:

The user account is set in the *User Settings* dialog box of the *Setup* OPUS pull-down menu.

8.5.18 GET_BENCH

Syntax:

“GET_BENCH”

Description:

Retrieves the configuration file of the currently selected spectrometer.

Return Value:

“OK”

Return Value 2:

<OpticsFile>

8.5.19 UPDATE_BENCH

Syntax:

“UPDATE_BENCH <OpticsFile>”

Description:

Triggers OPUS to initialize the optics configuration using the settings stored in the <OpticsFile>.

Return Value:

“OK” or error message.

Error:

“Syntax: UPDATE_BENCH <inifile>”

“RebuildParmText error”

8.5.20 COMMAND_SAY

Syntax:

"COMMAND_SAY <Text>"

Description:

Returns the transferred commands in text format.

Return Value:

<Text>

Comment:

This command serves to test the communication between OPUS and the client program. It can also be used to forward parameters to scripts. To do this, call the OpusCommand function of a form created with the OpenForm command (or selected with FormByName), and forward parameters using COMMAND_SAY <Parameter>. The form receives the parameter with OnOpusResult <Parameter>.

8.5.21 REPORT_INFO

Syntax:

"REPORT_INFO"

Description:

Retrieves information about the number of main and sub reports of an OPUS report block.

Return Value:

"OK" or error message.

Return Value 2:

<#Main reports>

<#Sub reports 1>

...

<#Sub reports N>

Error:

"No Filename or Filenumber defined"

“No Blocktype defined”

“Error Reading Report”

Comment:

First, the total number of main reports is returned, followed by the number of sub reports contained in each main report. Each line holds only one number.

The information is obtained from the OPUS report block selected by the READ_FROM_FILE, READ_FROM_ENTRY and READ_FROM_BLOCK commands.

8.5.22 HEADER_INFO

Syntax:

“HEADER_INFO <Main report> <Sub report>”

Description:

Returns the number of lines in an OPUS report block header.

Return Value:

“OK” or error message.

Return Value 2:

<Lines>

Error:

“No Filename or Filenumber defined”

“No Blocktype defined”

“Error Reading Report”

Comment:

If no sub report is specified, the number of lines in the header of the main report block is returned instead. If also no main report is specified, the first main report will be taken.

The information is obtained form the OPUS report block selected by the READ_FROM_FILE, READ_FROM_ENTRY and READ_FROM_BLOCK commands.

8.5.23 MATRIX_INFO

Syntax:

“MATRIX_INFO <Main report> <Sub report>”

Description:

Returns the dimension (number of rows and columns) of a matrix stored in an OPUS report block.

Return Value:

“OK” or error message.

Return Value 2:

<Rows>

<Columns>

Error:

“No Filename or Filenumber defined”

“No Blocktype defined”

“Error Reading Report”

Comment:

If no sub report is specified, the number of rows in the main reports’ header will be returned. If also no main report is specified, the first main report will be taken.

The information is obtained form the OPUS report block selected by the READ_FROM_FILE, READ_FROM_ENTRY, and READ_FROM_BLOCK commands.

8.5.24 MATRIX_ELEMENT

Syntax:

“MATRIX_ELEMENT <Main report> <Sub report> <Row> <Column>”

Description:

Reads an element from a data matrix of an OPUS report block. The index of the main/sub report as well as the index of the row and column has to be indicated.

Return Value:

"OK" or error message.

Return Value 2:

<MatrixElement>

Error:

"Syntax: MATRIX_ELEMENT <MainReport> <SubReport> <Row> <Column>"

"No Filename or Filenumber defined"

"No Blocktype defined"

"Error Reading Report"

Comment:

If the main report should be accessed, the sub report number must be set to "0".

Determine the total number of rows and columns, using the MATRIX_INFO command.

All values are converted to text format prior to the transfer, regardless of the data format of the element.

The information is obtained form the OPUS report block selected by the READ_FROM_FILE/READ_FROM_ENTRY, and READ_FROM_BLOCK commands.

8.5.25 HEADER_ELEMENT

Syntax:

"HEADER_ELEMENT <Main report> <Sub report> <Row>"

Description:

Reads an element from the OPUS report block header. The index of the main/ sub report as well as the number of the row has to be indicated.

Return Value:

"OK" or error message.

Return Value 2:

<ElementName>

<ElementContent>

Error:

“Syntax: HEADER_ELEMENT <MainReport> <SubReport> <Row>”

“No Filename or Filenumber defined”

“No Blocktype defined”

“Error Reading Report”

Comment:

The name of the feature in the selected header row and its value will be returned. If the main report should be accessed, the sub report number must be set to “0”.

Determine the total number of rows and columns using the HEADER_INFO command.

All values are converted to text format prior to the transfer, regardless of the data format of the element.

The information is obtained form the OPUS report block selected by the READ_FROM_FILE/READ_FROM_ENTRY and READ_FROM_BLOCK commands.

8.5.26 COMMAND_MODE

Syntax:

“COMMAND_MODE”

Description:

Sets the mode for processing a command line to COMMAND_MODE. This mode runs commands and programs in the background and returns a message after termination of the program.

Return Value:

“OK”

Comment:

Usually, this mode doesn’t need to be explicitly set, since these modi are pre-defined for the different transfer types and interfaces or alternatively are set by different calls (like OpusExecute).

8.5.27 EXECUTE_MODE

Syntax:

"EXECUTE_MODE"

Description:

Sets the mode for processing a command line to EXECUTE_MODE. This mode runs commands and programs in the background, but does not wait for the programs to terminate. No message will be returned when a program has finished.

Return Value:

"OK"

Comment:

Usually, this mode doesn't need to be explicitly set, since these modi are pre-defined for the different transfer types and interfaces or alternatively are set by different calls (like OpusExecute).

8.5.28 REQUEST_MODE

Syntax:

"REQUEST_MODE"

Description:

Sets the mode for processing a command line to REQUEST_MODE. This mode does not run commands and programs in the background, but waits for the programs to terminate. The result will be returned as soon as the program terminates.

Return Value:

"OK"

Comment:

Usually, this mode doesn't need to be explicitly set, since these modi are pre-defined for the different transfer types and interfaces or alternatively are set by different calls (like OpusExecute).

8.5.29 CLOSE_OPUS

Syntax:

“CLOSE_OPUS”

Description:

Terminates OPUS.

Return Value:

No return values.

Comment:

This operation is similar to closing the OPUS user interface window.

8.5.30 TAKE_REFERENCE

Syntax:

“TAKE_REFERENCE <Experiment file>”

Description:

Performs a reference measurement using the specified <Experiment file>.

Return Value:

“OK” or error message.

Error:

“Error in Opus Command Line Execution - ID: %d”

8.5.31 MEASURE_SAMPLE

Syntax:

“MEASURE_SAMPLE <Experiment file>”

Description:

Performs a sample measurement using the specified <Experiment file> and returns the acquired spectral data as text.

Return Value:

“OK” or error message.

Return Value 2:

Result File:<File number>

<File name>

Block: <Block type>

<UnitsX>

<UnitsY>

Points: <Number of points>

<x1> <y1>

...

<xn> <yn>

Error:

“Error in Opus Command Line Execution - ID: %d”

Comment:

All blocks of the new file (specified by the experiment file) are transmitted in succession as data point tables.

8.5.32 COMMAND_LINE

Syntax:

“COMMAND_LINE <Command line>”

Description:

Calls an OPUS function as command lines.

Return Value:

“OK” or error message.

Return Value 2:

Only in combination with COMMAND_MODE

<ThreadID>

Error:

“Error in Opus Command Line Execution - ID: %d”

Comment:

In this exception, the keyword COMMAND_LINE can be omitted, because OPUS tries to interpret all unknown C/S commands in command line notation.

The actual type of command processing depends on the call of the command (in case of scripts for example OpusExecute), or the settings made by COMMAND_MODE, EXECUTE_MODE, and REQUEST_MODE.

If COMMAND_MODE was selected, an identification number is supplied for the background calculation, which can be used to abort the function in case of STOP_THREAD.

8.5.33 STOP_THREAD

Syntax:

"STOP_THREAD <ThreadID>"

Description:

Terminates a OPUS processing function which was started by the COMMAND_LINE function while COMMAND_MODE was selected.

Return Value:

"OK" or error message.

Error:

"Syntax: STOP_THREAD <ThreadID>"

Comment:

In COMMAND_MODE, COMMAND_LINE starts the function in the background and returns an identification number. This number can be used to abort the function. This is similar to the *Abort Task* command of the task manager.

Note: Aborting a program may result in data loss and produce corrupt OPUS files. Therefore, it should only be used in emergencies.

8.5.34 ACTIVATE_DIALOG

Syntax:

"ACTIVATE_DIALOG <Command line>"

Description:

Starts the dialog box of an OPUS function.

Return Value:

"OK" or error message.

Error:

"Syntax: ACTIVATE_DIALOG CommandLine()"

Comment:

Opening an OPUS function dialog box within another program usually is not very practical, since the program cannot control the dialog box once it has been opened. A command line is required as a parameter similar to direct command processing.

8.5.35 LOAD_EXPERIMENT

Syntax:

"LOAD_EXPERIMENT <Experiment file>"

Description:

Loads an experiment file in OPUS and sets the parameters for subsequent data acquisitions.

Return Value:

"OK" or error message.

Error:

"Syntax: LOAD_EXPERIMENT <parameter file>"

"Unable to load Experiment file"

Comment:

This command is similar to the respective function of the OPUS *Measurement* dialog box.

8.5.36 GET_USERRIGHTS

Syntax:

"GET_USERRIGHTS"

Description:

Retrieves the rights of the current user.

Return Value:

"OK"

Return Value2:

A list of user rights separated by semicolons or "No Rights"

Comment:

Allows to adjust programs/scripts to perform different actions, depending on the user rights.

8.5.37 PACKET_AVAILABLE

Syntax:

"PACKET_AVAILABLE <Packet name>"

Description:

Tests if certain OPUS software packages are installed on a computer.

Return Value:

"Yes", "No" or error message.

Error:

"Syntax: PACKET_AVAILABLE <Packet name>"

Comment:

Allows a script or program to determine, whether a software package or an OPUS function is available at all. This applies to QUANT, SEARCH, 3D etc..

8.5.38 GET_CLIENTAREA

Syntax:

"GET_CLIENTAREA"

Description:

Retrieves the available window size of the OPUS main window. This is dependent on the chosen screen resolution. The result can be used for the positioning of script forms and spectrum windows etc.

Return Value:

"OK"

Return Value2:

<width> <height>

Comment:

The returned values can be used as parameters for POSITION_WINDOW.

8.5.39 ACTIVATE_DISPLAY

Syntax:

“ACTIVATE_DISPLAY” <WindowID>

Description:

A spectrum window can be activated using this command. It will then be displayed in the front. The window specified by the ID number will then be the active window for displaying the spectra.

Return Value:

“OK” or error message

Error:

“Syntax: ACTIVATE_DISPLAY <window>

Comment:

If new files are loaded or created by other OPUS functions, they will then be displayed in the currently active window. The active window can be determined with this function. Whereas SET_WINDOW is only valid for files used in script, here the active window e.g. for manual loading can be set.

8.5.40 GET_LIMITS

Syntax:

“GET_LIMITS <WindowID>”

Description:

Lists the actual display limits of the window.

Return Value:

“OK” or error message

Return Value 2:

<X1> <Y1> <X2> <Y2>

Error:

“Syntax: GET_LIMITS <window>”

Comment:

The <WindowID> can either be a result of NEW_WINDOW or GET_DISPLAY

8.5.41 SET_LIMITS

Syntax:

“SET_LIMITS <WindowID> <X-start> <X-end> <Y-start> <Y-end>”

Description:

Sets the display limits of the window to the given values. this is useful to e.g. enlarge certain areas of the spectrum automatically. The four values determine the coordinates for the new display limits.

Return Value:

“OK” or error message

Error:

“Syntax: SET_LIMITS <window> <xsp> <xep> <ymn> <ymx>”

Comment:

The <WindowID> can either be a result of NEW_WINDOW or GET_DISPLAY

8.5.42 DISPLAY_BLOCK

Syntax:

“DISPLAY_BLOCK <WindowID> <color>”

Description:

Displays a datablock of an OPUS file selected by the commands READ_FROM_ENTRY, READ_FROM_FILE or READ_FROM_BLOCK in a display window determined by <windowID>. <color> determines the color of the curve as RGB value.

Return Value:

“OK” or error message

Error:

“No Filename or Filenumber defined”

“No Blocktype defined”

“Syntax: DISPLAY_BLOCK <window> <color>”

Comment:

The <WindowID> can either be a result of NEW_WINDOW or GET_DISPLAY

8.5.43 UNDISPLAY_BLOCK

Syntax:

“UNDISPLAY_BLOCK <WindowID>”

Description:

Removes a datablock of an OPUS file specified by READ_FROM_ENTRY, READ_FROM_FILE or READ_FROM_BLOCK from the window identified by <WindowID>.

Return Value:

“OK” or error message

Error:

“No Filename or Filenumber defined”

“No Blocktype defined”

“Syntax: UNDISPLAY_BLOCK <window>”

Comment:

The <WindowID> can either be a result of NEW_WINDOW or GET_DISPLAY

8.5.44 ENUM_STRINGS

Syntax:

“ENUM_STRINGS <parametername>

Description:

Possible values for a parameter of type ENUM can be requested at the given conditions, e.g. depending on the spectrometer.

Return Value:

"OK" or error message

Return Value 2:

<number of the following valid strings>

<first parameterstring>

...

<last parameterstring>

Error:

"Syntax: ENUM_STRINGS <parameter name>"

"Invalid Parameter Name"

"No Enum Strings"

Comment:

The parameter name transferred as an argumentis a three letter abbrevation of a parameter.

8.5.45 GET_VERSION

Syntax:

"GET_VERSION>"

Description:

Returns the version of the currently running OPUS NT program.

Return Value:

"OK"

Return Value 2:

<Version>

Error:

Comment:

Enables the reaction on and the controlling of, different current OPUS versions from *one* program.

8.5.46 ASK_THREAD

Syntax:

“ASK_THREAD <ProcessID> <special command>

Description:

Enables the interprocess communication of an external program with a running Opus function.

Return Value:

“OK” or error message

Return Value 2:

depending on the transferred command

Error:

“ASK_THREAD failed”

“Invalid Thread ID”

Comment:

This direct communication with currently running OPUS functions is intended only for very special applications. It is mentioned here only for the sake of completeness. However it is actually reserved to internal programming and is used for the coupling with other instruments. One receives the ProcessID either when starting the function in the COMMAND_MODE or through FIND_FUNCTION

8.5.47 FIND_FUNCTION

Syntax:

“FIND_FUNCTION <function name>”

Description:

Determines whether a certain OPUS function is executed in the background. The returned ID can be used to stop the function or to communicate with it (if supported).

Return Value:

“OK” or error message

Return Value 2:

<ProcessID> for identification

Error:

“Syntax: FIND_FUNCTION <FunctionName> or <ThreadID>

“Function not found”

Comment:

This direct communication with currently running OPUS functions is intended only for very special applications. It is mentioned here only for the sake of completeness. However it is actually reserved to internal programming and is used for the coupling with other instruments.

8.5.48 WORKBOOK_MODE

Syntax:

“WORKBOOK_MODE ON|OFF”

Description:

Turns the tabs for switching between different windows at the bottom of the OPUS window on or off.

Return Value:

“OK” or ON|OFF”

Error:

Comment:

When the buttons are deactivated, switching between different windows is no longer possible. In the case that a simple user interface is required, one prevents thereby deviations from the operational sequence intended.

8.5.49 GET_SELECTED

Syntax:

“GET_SELECTED”

Description:

Supplies the names of the selected (red bordered) files.

Return Value:

“OK” or error message

Return Value 2:

<name of selected files>

Error:

“Error while getting file info”

Comment:

The normal behavior of OPUS, to automatically select marked files for the processing functions, is extended to self-written extension functions.

8.5.50 LIST_BLOCKS

Syntax:

“LIST_BLOCKS”

Description:

Lists all available spectral data blocks of the OPUS file delivered by the command READ_FROM_ENTRY or READ_FROM_FILE.

Return Value:

“OK” or error message

Return Value 2:

<number of block names>

<first block name>

...

<last block name>

Error:

“No Filename or Filenumber defined”

“Error getting blocks”

Comment:

Enables to determine which blocks are containing an unknown file and then work with the correct ones accordingly.

8.5.51 SHOW_TOOLBAR

Syntax:

“SHOW_TOOLBAR <toolbar>”

Description:

Shows a toolbar. Valid parameters are:

MENU, STANDARD, COMMANDLINE, PLE, DISPLAY, MEASURE, MANIPULATE, EVALUATE, MDISPLAY, PLOT_PRINT, MACRO, INFO, USER, SETUP, FILE, BROWSER, STATUSBAR

Return Value:

“OK” or “Already visible”

Error:

“Syntax: SHOW_TOOLBAR <toolbarID>”

“Unknown Toolbar”

Comment:

Warning: The adjustments on the desktop made with this function are stored when leaving OPUS. In order to avoid unwanted effects, all modifications of the original configuration should be cancelled again before the final termination of the self-written program!

8.5.52 HIDE_TOOLBAR

Syntax:

“HIDE_TOOLBAR <toolbar>”

Description:

Hides a toolbar. Valid parameters are:

MENU, STANDARD, COMMANDLINE, PLE, DISPLAY, MEASURE, MANIPULATE, EVALUATE, MDISPLAY, PLOT_PRINT, MACRO, INFO, USER, SETUP, FILE, BROWSER, STATUSBAR

Return Value:

“OK” or “Already visible”

Error:

“Syntax: HIDE_TOOLBAR <toolbarID>”

“Unknown Toolbar”

Comment:

Warning: The adjustments on the desktop made with this function are stored when leaving OPUS. In order to avoid unwanted effects, all modifications of the original configuration should be cancelled again before the final termination of the self-written program!

8.5.53 QUICK_PRINT

Syntax:

"QUICK_PRINT"

Description:

Activates the function "Quickprint". The currently active window will be printed.

Return Value:

"OK"

Error:

Comment:

To print a certain window with this function, it has to be activated with ACTIVATE_DISPLAY first.

9 Problems and Solutions

9.1 Startup problem

The only two things that may go wrong with InitInstrument are that the .nti file is not found in the application directory (or it is corrupted) or that there is no userdatabase subdirectory with a valid userdatabase.dat file. Take it from an opus installation or try the one on the CD.

The next thing in the order of the startup would be loading of the workspace:

9.2 "Failed to open document"

The message "Failed to open document" refers to an opus workspace file that is also used by the DLL. If there is no default.ows in the directory of the project or this is corrupted you will get this message and you cannot continue at all from there on.

Please try to copy the file default.ows from the original CD or you can also use an unmodified version from a normal Opus installation.

9.3 File not found in filetable

The most likely cause for a file not found in filetable is that the result file could not be created. Try to add NAM and PTH also to the command and make sure that the destination is not the problem. Apart from that in case you have corrupted the opus.bg file you might have to delete that file.

9.4 Error in commandline

The command line you are sending use very specific syntax and you have to make sure that all the characters are there. If something could not be interpreted you might get errors like this. The Error in Opus Command Line Execution basically informs us that Opus received a command it did not recognize.

The error code 8 means COMMAND_INVALID_FILES, i.e. something is wrong with the file or filelist you are providing. It may be missing or not containing the right blocks. You should add the right blocktype in the command line.

The ID:2 in this case is "unknown command". Thus there must be something wrong with the command or the parameters passed to the opus function here. The error message is the result of specific function call and you must check the exact! parameters used.

9.5 VB error 48/53

This is something like DLL not found To avoid the error 48 VB has to find all the DLLs. You have to have all the dlls and support files within the directory of your application.