TopSpin 2.1
Version 2.1.2

AU Programming

think forward
## Contents

### Chapter 1 Introduction

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>What is new in TOPSPIN 2.1.</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>What is new in TOPSPIN 2.0.</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>What are AU programs?</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>Other Manuals describing AU Programs/Macros</td>
<td>6</td>
</tr>
<tr>
<td>1.5</td>
<td>Quick reference to using AU programs</td>
<td>7</td>
</tr>
<tr>
<td>1.6</td>
<td>Installing and compiling AU programs</td>
<td>7</td>
</tr>
<tr>
<td>1.7</td>
<td>Executing AU programs</td>
<td>8</td>
</tr>
<tr>
<td>1.8</td>
<td>Viewing AU programs</td>
<td>8</td>
</tr>
<tr>
<td>1.9</td>
<td>About AU macros</td>
<td>9</td>
</tr>
<tr>
<td>1.10</td>
<td>About Bruker library functions</td>
<td>9</td>
</tr>
<tr>
<td>1.11</td>
<td>Creating your own AU programs</td>
<td>10</td>
</tr>
<tr>
<td>1.12</td>
<td>How an AU program is translated into C-code</td>
<td>16</td>
</tr>
<tr>
<td>1.13</td>
<td>Listing of all predefined C-statements</td>
<td>19</td>
</tr>
</tbody>
</table>

### Chapter 2 Inventory of AU macros and Bruker library functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Naming conventions</td>
<td>23</td>
</tr>
<tr>
<td>2.2</td>
<td>Macros for dataset handling</td>
<td>25</td>
</tr>
<tr>
<td>2.3</td>
<td>Macros prompting the user for input</td>
<td>27</td>
</tr>
<tr>
<td>2.4</td>
<td>Macros handling Topspin parameters</td>
<td>27</td>
</tr>
<tr>
<td>2.5</td>
<td>Acquisition macros</td>
<td>29</td>
</tr>
<tr>
<td>2.6</td>
<td>Macros handling the shim unit and the sample changer</td>
<td>29</td>
</tr>
<tr>
<td>2.7</td>
<td>Macros handling the temperature unit</td>
<td>31</td>
</tr>
<tr>
<td>2.8</td>
<td>Macros handling the MAS unit</td>
<td>31</td>
</tr>
<tr>
<td>2.9</td>
<td>1D processing macros</td>
<td>33</td>
</tr>
<tr>
<td>2.10</td>
<td>Peak picking, integration and miscellaneous macros</td>
<td>35</td>
</tr>
<tr>
<td>2.11</td>
<td>Macros for algebraic operations on datasets</td>
<td>35</td>
</tr>
<tr>
<td>2.12</td>
<td>Deconvolution macros</td>
<td>37</td>
</tr>
<tr>
<td>2.13</td>
<td>2D processing macros</td>
<td>37</td>
</tr>
<tr>
<td>2.14</td>
<td>Macros reading and writing projections etc.</td>
<td>40</td>
</tr>
<tr>
<td>2.15</td>
<td>3D processing macros</td>
<td>41</td>
</tr>
<tr>
<td>2.16</td>
<td>Spectral Width calculation macros</td>
<td>42</td>
</tr>
<tr>
<td>2.17</td>
<td>Plot editor related macros</td>
<td>42</td>
</tr>
<tr>
<td>2.18</td>
<td>Macros converting datasets</td>
<td>44</td>
</tr>
<tr>
<td>2.19</td>
<td>Macros to execute other AU programs, Topspin macros or commands</td>
<td>44</td>
</tr>
<tr>
<td>2.20</td>
<td>Bruker library functions</td>
<td>45</td>
</tr>
<tr>
<td>2.21</td>
<td>Macros for loop control</td>
<td>46</td>
</tr>
<tr>
<td>2.22</td>
<td>Macros to return from an AU program</td>
<td>46</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3</td>
<td>General AU macros</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>TOPSPIN Interface functions</td>
<td>57</td>
</tr>
<tr>
<td>5</td>
<td>Macros changing the current AU dataset</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>Macros copying datasets</td>
<td>73</td>
</tr>
<tr>
<td>7</td>
<td>Macros handling rows/columns</td>
<td>77</td>
</tr>
<tr>
<td>8</td>
<td>Macros converting datasets</td>
<td>89</td>
</tr>
<tr>
<td>9</td>
<td>Macros handling TOPSPIN parameters</td>
<td>95</td>
</tr>
<tr>
<td>10</td>
<td>Macros for Plot Editor/autoplot</td>
<td>109</td>
</tr>
<tr>
<td>11</td>
<td>Macros prompting the user for input</td>
<td>119</td>
</tr>
<tr>
<td>12</td>
<td>Bruker library functions</td>
<td>125</td>
</tr>
<tr>
<td>13</td>
<td>List of Bruker AU programs</td>
<td>157</td>
</tr>
<tr>
<td>14</td>
<td>TOPSPIN parameter types</td>
<td>175</td>
</tr>
<tr>
<td>14.1</td>
<td>Integer parameters</td>
<td>176</td>
</tr>
<tr>
<td>14.2</td>
<td>Float parameters</td>
<td>177</td>
</tr>
<tr>
<td>14.3</td>
<td>Double parameters</td>
<td>178</td>
</tr>
<tr>
<td>14.4</td>
<td>Character-string parameters</td>
<td>178</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 What is new in TOPSPIN 2.1

Changes in TOPSPIN 2.1 with respect to AU programs.

1. The macro XAU requires two arguments, allowing you to freely choose the arguments to be propagated from the calling AU program.

2. C-language argument syntax iarg_v and iarg_c can be used in AU programs.

3. New AU macros to delete data have been added: DELETEPROCDATA, DELETEIMAGINARYDATA, DELETERAWDATA, DELETEPROCNO, DELETEEXPNO, DELETENAME.

4. New AU macros to fetch/store nD data have been added: FETCHPARN, FETCHPARNs, STOREPARN, STOREPARNs.

5. The functions `getParfileDirForRead` and `getParfileDirForWrite` replace the functions `getstan` and `PathXWinNMR*`.

1.2 What is new in TOPSPIN 2.0

Changes in TOPSPIN 2.0 with respect to AU programs.
1. AU programs that contain a plotting command can be entered with the argument `noplot`. This argument prevents plotting.

2. All AU-macros, e.g. EF, APK, QUIT must be specified in capital letters. In previous versions of TOPSPIN and its predecessor XWIN-NMR, capital letters were recommended but not required.

3. New macros exist for automatic creation of Plot Editor layouts. Examples are LAYOUT_OBJ_1D and LAYOUT_ADD.

### 1.3 What are AU programs?

AU programs can be considered as user defined TOPSPIN commands. Any repetitive task is most effectively accomplished through an AU program. All commands which can be entered on the TOPSPIN command line can also be entered in an AU program in the form of macros. This includes selecting and changing datasets, reading and setting parameters, starting acquisitions, processing data and plotting the result. A simple AU program is nothing else than a sequence of such macros which execute the corresponding TOPSPIN commands. However, AU programs may also contain C-language statements. In fact, an AU program is a C-program because all AU macros are translated to C-statements. TOPSPIN automatically compiles AU programs to executable binaries, using a C-compiler.

TOPSPIN offers three other ways of creating user defined commands: TOPSPIN macros (not to be confused with AU macros), Tcl/Tk scripts and Python programs. They differ from AU programs in that they do not need to be compiled.

### 1.4 Other Manuals describing AU Programs/Macros

Creating and using AU programs is described and referred to in various other manuals:

- **Processing Reference Guide**: for each processing command for which an AU macro exists, this macro and its usage is specified.

- **Acquisition Reference Guide**: for each acquisition command for which an AU macro exists, this macro and its usage is specified.
• **NMR Guide**: AU programs can be sorted and listed according to their usage showing their names and short descriptions.
• **Plot Editor manual**: chapter about AU program macros for plotting.

### 1.5 Quick reference to using AU programs

Bruker delivers a library of standard AU programs with TOPSPIN. After TOPSPIN has been installed you must do the following in order to use them:

1. Run `expinstall` once to install all AU programs
2. Run `compileall` once to compile all AU programs
3. Enter the name of an AU program to execute it

Furthermore, you can write your own AU programs in the following way:

1. Enter `edau <name>`
   The file `<name>` will be opened with a text editor
2. Do one of the following:
   • Write your own AU program from scratch
   • Read in an existing AU program and modify it according to your needs
3. Click `Save, exit and compile`.
4. Enter the name of the AU program to execute it.

After you have installed a new version of TOPSPIN, you must run `expinstall` and `compileall` again to install and compile both Bruker’s and your own AU programs.

### 1.6 Installing and compiling AU programs

When you have installed a new version of TOPSPIN, you must install the library AU programs once by executing the TOPSPIN command `expinstall`.

---

1. If you are not using the internal editor, you have to compile the AU program in a separate step with the command `cplbruk <name>`. 

Your own AU programs which you created under a previous version of TOPSPIN are still available, they only need to be re-compiled.

An AU program is automatically compiled, the first time it is executed, i.e. when its name is entered on the command line.

To compile an AU program without executing it:
- enter \texttt{cplbruk \langle name\rangle}
- or
- enter \texttt{edau \langle name\rangle} and click \textit{Exit and Compile}.

To compile all Bruker AU programs:
- enter \texttt{compileall}

\section*{1.7 Executing AU programs}

Once an AU program has been installed, there are 3 different ways to execute it:

1. Enter the name of the AU program. This will work if:
   - The AU program is already compiled
   - No TOPSPIN command or macro \(^1\) with the same name exist

2. Enter \texttt{edau}

   A list of available AU programs will appear. Click on the AU program you want to execute and click \textit{Execute}.

\section*{1.8 Viewing AU programs}

You can view existing AU programs in the following

a) Enter \texttt{edau}

   A dialog box listing all AU programs is opened. From the \textit{Options} menu, you can choose to display \textit{Bruker defined}, \textit{User defined} or \textit{All AU}

---

1. Here we refer to a TOPSPIN macro created with \texttt{edmac}
programs.

b) Click on an AU program in the list

When you select a Bruker AU program, it is shown in view mode which means you cannot edit it. When you click on a user-defined AU program it is shown in edit mode which means you can change it.

3. Enter listall

A list and a short description of all library AU programs is stored in the file listall in the users home directory. Note that this list is also available in Chapter 13 of this manual.

1.9 About AU macros

We will use the word macro rather often throughout this manual referring to AU macros. This should not be confused with TOPSPIN macros which are files containing a sequence of TOPSPIN commands. TOPSPIN macros are created with edmac and executed with xmac. An AU macro, however, is a statement in an AU program which defines one or more TOPSPIN commands, library functions or C-language statements. In its simplest form, an AU macro defines one TOPSPIN command. For example the macros ZG and FT execute the TOPSPIN commands zg and ft, respectively. Other macros like FETCHPAR and IEXPNO do not define TOPSPIN commands, their function is only relevant in the context of an AU program. More complex macros may contain several TOPSPIN commands and/or C-statements. All macros in AU programs should be written in capital letters. They are automatically translated to the corresponding C-code when the AU program is compiled. AU macros are defined in the file:

/tshome/prog/include/aucmd.h

1.10 About Bruker library functions

Bruker library functions are C-functions which are contained in Bruker libraries. They offer several features which are also used in the TOPSPIN interface, for example the display of a list of datasets from which the user can select one dataset. If you use a Bruker library function in an AU program the corresponding library is automatically linked to the AU program
during compilation. The most important and versatile Bruker library functions are described in chapter 9.

1.11 Creating your own AU programs

1.11.1 Writing a simple AU program

Before you start writing an AU program, you might want to check if an AU program already exists which (almost) meets your requirements. If this is not the case, you can write your own AU program in the following way:

1. Enter `edau <au-name>`
   Your preferred TOPSPIN text editor will be opened 1

2. Do one of the following:
   - Insert an existing library AU program and modify it to your needs.
   - Write a new AU program using the macros as described in this manual.

   The last macro in an AU program should always be QUIT (or QUITMSG).

3. Click `Save, exit and compile` 2.

1.11.2 Using variables

Since AU programs are C programs you can use C-language variables. Several variables are already predefined for usage in AU programs. In fact, we distinguish three different types of variables: predefined dedicated variables, predefined general variables and user defined variables.

1.11.2.1 Predefined dedicated variables

Predefined dedicated variables have the following properties:
   - they do not need to be declared in an AU program

---

1. To change the TOPSPIN text editor enter `set` and click `Miscellaneous`
2. If you are not using the internal editor, you have to compile the AU program in a separate step with the command `cplbruk <name>`
• their declaration is automatically added during compilation
• they are known to the AU main body and to possible subroutines
• they are set implicitly by certain macros, e.g. the variable expno is set by macros like DATASET and IEXPNO
• they should not be set explicitly, so do NOT use statements like:

```
expno = 11;
FETCHPAR("NS", &expno)
```
• they can be evaluated in macros or C-statements, e.g.:

```
DATASET(name, expno, 2, disk, "guest")
i1=expno+1;
```
• examples of different types of predefined dedicated variables are:
  char-string: name, disk, user, name2
  integer: expno, procno, loopcount1, loopcount2, lastparflag

A complete list of all predefined dedicated variables with their types can be found in Chapter 1.13.2

1.11.2.2 Predefined general variables

Predefined general variables have the following properties:
• they do not need to be declared in an AU program
• their declaration is automatically added during compilation
• they are known to the AU main body but not to possible subroutines
• they can be freely used for various purposes
• examples of different types of predefined general variables are:

```
integer: i1, i2, i3
float: f1, f2, f3
double:d1, d2, d3
char-string: text
```

A complete list of all predefined general variables with their types and initial values can be found in Chapter 1.13.3.
1.11.2.3 User defined variables

For simple AU programs the number of predefined general variables is sufficient, you do not need to declare any additional variables. For more complex AU programs you might need more variables or you might want to use specific names. In these cases you can define your own variables in the AU program. User defined variables have the following properties:

- they must be declared at the beginning of an AU program
- they can be freely used for various purposes
- they are known to the main AU program but not to possible subroutines
- examples of declarations are:

  int ivar1, ivar2;
  float fvar1, fvar2, fvar3;
  double dvar1, dvar2, dvar3;
  char cstr1[20], cstr2[200];

1.11.3 Using AU macros with arguments

Several AU macros take one or more arguments. Arguments can be constants (values) or variables. In fact, an argument can be specified in four different ways as described here for the macro REXPNO:

- as a constant, e.g.:
  
  REXPNO(3)

- as a predefined dedicated variable e.g.:

  REXPNO(expno+1)

- as predefined general variable, e.g.:

  i1=6;
  REXPNO(i1)

- as a user defined variable, e.g.:

  int my_exp;
  ....
  my_exp=1;
  REXPNO(my_exp)
It is very important that the arguments are of the correct type. Macros can take arguments of the type integer (like \texttt{REXPNO}), float, double or character-string.

Some macros, for example \texttt{STOREPAR}, take \texttt{TOPSPIN} parameters as arguments and each parameter is of a certain type. For example, the AU statement

\begin{verbatim}
STOREPAR("O1", d1)
\end{verbatim}

stores the value of the variable \(d1\) into the parameter \(O1\). The predefined (double) variable \(d1\) is used since \(O1\) is of the type double. The second argument could also be a constant, e.g.:

\begin{verbatim}
STOREPAR("O1", 287.15)
\end{verbatim}

A list of all \texttt{TOPSPIN} parameters and their type can be found in Chapter 14.

### 1.11.4 Using AU programs with arguments

An AU program can be used with arguments. Arguments are available within the AU program as C-languages variables:

\begin{verbatim}
   i_argc: the number of arguments  
i_argv: the arguments  
   cmd: all specified arguments concatenated
\end{verbatim}

The first argument is the AU program pathname and the second argument always \texttt{exec}. So for an AU program entered without arguments, \(i\_argc = 2\) and \(cmd\) is an empty string. For the example \texttt{myau a1 a2}

\begin{verbatim}
i_argc = 4
i_argv[0] = myau
i_argv[1] = exec
i_argv[2] = a1
i_argv[3] = a2
cmd = "a1 a2"
\end{verbatim}

Note that \texttt{cmd} is actually legacy code whose usage is discouraged. It may no longer be supported in future versions.
1.11.5 Using C-language statements

AU programs can contain AU macros but also C-language statements like:

- define statements, e.g.: #define MAXSIZE 32768
- include statements, e.g.: #include <time.h>
- variable declarations, e.g. int ivar;
- variable assignments, e.g.: ivar = 20;
- loop structures, e.g.: for, while, do
- control structures, e.g.: if-else
- C-functions, e.g.: strcpy, strcmp, sprintf

Important: several C-language statements (including declarations of variables) are already predefined and automatically added during compilation of the AU program.

An example of an AU program using macros and C-statements is:

```c
int eno, pno;
char datapath[500], dataname[50], datauser[50], datadisk[200];

(void) strcpy (dataname,name);
(void) strcpy (datauser,user);
(void) strcpy (datadisk,disk);
eno = expno;
pno = procnos;
(void) sprintf (datapath,"%s/data/%s/nmr/%s/%d/pdata/%d/title",
  datadisk, datauser, dataname, eno, pno);
if ( (i1 = showfile (datapath)) < 0 )
{
    Proc_err (DEF_ERR_OPT,"Problems with showfile function");
}
QUIT
```

Note that QUIT is an AU macro, strcpy and sprintf are C-functions and showfile and Proc_err are Bruker library functions.

For an explanation of C-functions and more information on C-language we refer to the literature on C-programming.
1.11.6 Additional hints on C-statements

If you are using C-language code in your AU programs, then there are a few things to be considered.

1. Using C-language header files

Several C-language header files are automatically included in your AU program during compilation. If you are using C-code which requires additional header files you must write your AU program in a special way. The main AU program should be a call to a subroutine which performs the actual task of the AU program. The include statements for the header file must be entered between the main AU program and the subroutine. This gives the following structure:

```c
subroutine(curdat, cmd)
QUIT
#include <headerfile.h>
subroutine(curdat, cmd)
char *curdat, *cmd;
{
    MACRO1
    MACRO2
}
```

Such a structure is used in several Bruker library AU programs (e.g. amplstab, decon_t1, etc.). Several Bruker library functions like `PrintExpTime`, `gethighest`, `getxwinvers`, `pow_next` and `unlinkpr` also require an include statement in the AU program (see Chapter 12).

2. Some macros, e.g. IEXPNO and IPROCNO change the current AU dataset but do not make it available for subsequent commands. If they are followed by a CPR_exec or any C-statement which access the current AU dataset, then you must precede that statement with `SETCURDATA` (see also the descriptions of `SETCURDATA`, IEXPNO etc. in Chapter 5).

3. If you are using C-languages loop statements like `for`, `do` or `while` or control statements like `if`, we strongly recommend to always put the body of such statements between `{}`. If the body only contains simple macros like ZG or FT you can omit them because these macro definitions already contain `{}`. However, more complex macros might internally
define C-statements that include loop or control structures. If such a macro is used within a loop or control structure in the AU program, then you create nested loops which require the usage of {}.

1.11.7 Viewing Bruker standard AU programs for macro syntax

The syntax of many AU macros is trivial, just enter the TOPSPIN command in capital letters. Other macros and especially Bruker library functions are more complex. A detailed description of frequently used AU macros and functions can be found in subsequent chapters of this manual. Alternatively, you can also look for an existing AU program containing this macro or function. If, for example, you want to know the syntax of the macro WRPA, just search for an AU program containing the text WRPA in the directory:

<ts_home>\prog\au\src.exam

using the Windows of Linux Search function.

1.12 How an AU program is translated into C-code

This paragraph is intended for users who want to get a deeper understanding of the compilation process. If you simply want to write and use AU programs you can skip this paragraph.

TOPSPIN automatically translates your AU program into C-language and compiles it. Files and directories used during AU program compilation are:

/<tshome>/exp/stan/nmr/au/makeau
/<tshome>/exp/stan/nmr/au/vorspann
/<tshome>/prog/include/aucmd.h
/<tshome>/prog/include/inc

The compilation process is entirely controlled by the script makeau which performs the following steps.

1. The file vorspann is concatenated with your AU program. This file contains a variety of definitions including
   • the C-program main statement
   • #include statements of C-header files (which in turn contain other definitions)
- `#define` statements which define constants
- predefined dedicated variables, e.g.: `name, disk, user, expno, procno`
- predefined general variables, e.g.: `text, i1, i2, i3, f1, f2, f3, d1, d2, d3`

2. After `vorspann` and your code have been concatenated, all macro definitions are replaced according to their definitions as described in the file `aucmd.h` and in the `inc` directory. In some cases, the name of the macro is the name of one of the files in `inc` directory and the entire content of the file represents that macro.

3. Step 2. results in a C program source file which this file is compiled and an executable program is created. By default, the compilation is done with the GNU C-compiler `gcc` which is delivered with TOPSPIN. The linking process is done with the native linker which is part of the native C-compiler `cc`. All AU program’s source files reside in:

   `/<tshome>/exp/stan/nmr/au/src`

   executables will be stored into:

   `/<tshome>/prog/au/bin`

The following section shows the result of concatenating `vorspann` with the following AU program:

```
EFP
APK
SREF
QUIT
```

For better presentation, only a part of `vorspann` is shown. All variables declared in `vorspann` are listed in chapter 1.10.

```
#include <stdio.h>
#include <stdlib.h>

...........................

main(argc,argv)
int argc;
char **argv;
{
char curdat[PATH_MAX];
char arglist[BUFSIZ];
```
int modret;
...
:void)getcurdat(1, curdat, disk, user, type, name, &expno, &procno);
...
modret = AU_program(curdat, arglist);
}

............................

AU_program(curdat, cmd)
char *curdat;
char *cmd;
{
  int i1=0, i2=0, i3=0;
  float f1=0, f2=0, f3=0, f998=0, f999=0;
  double d1=0, d2=0, d3=0;
  char text[BUFSIZ/2];

  GETCURDATA
  EFP
  APK
  SREF
  QUIT

  Note that the macro QUIT defines the closing C-language ‘}’ statement.

1.12.1 Using the native gcc compiler

By default, AU programs are compiled with the Bruker delivered gcc compiler. If you want to use the native operating system compiler, you can do that as follows.

1. From the Windows Explorer or LINUX file manager

   Open the following file with a text editor:

   <tshome>/exp/stan/nmr/au/makeau

2. Search for the following line:

   # $opt_native = 1;

   and remove the # character at the beginning of the line
3. Save and close the file.

4. Start TOPSPIN and compile your AU programs

Now, under Windows, the Visual C++ compiler will be used. Note that this is not a part of the standard operating system. Under LINUX, the default system GCC compiler is used.

To activate the native compiler for the current TOPSPIN session only, enter the following command:

```
env set DEBUG_MAKEAU=-native
```

### 1.13 Listing of all predefined C-statements

#### 1.13.1 Including header files

The following C-language header files are automatically included during compilation:

- `stdio.h, stdlib.h, unistd.h, string.h, errno.h, math.h, limits.h, fcntl.h`

which reside in the following directory:

- Under Windows: `<tshome>/GNU/usr/include`
- Under LINUX: `/usr/include`

and

- `erropt.h, brukdef.h, lib/uni.h, lib/libcb.h, lib/util.h, sample.h, aucmd.h`

which reside in the directory:

- `/tshome/prog/include`

Note that under LINUX, the packages `glibc-kernheaders` and `glibc-devel` must be installed to be able to compile AU programs (see Installation Guide Linux).

#### 1.13.2 Predefined dedicated variables

The following list contains all predefined dedicated variables, their type and the AU macros by which they are set. Note that most variables are set or modified by several macros and only one or two are listed here.
<table>
<thead>
<tr>
<th>type</th>
<th>variable</th>
<th>set by macros</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>lastparflag</td>
<td>USELASTPARS, USECURPARS</td>
</tr>
<tr>
<td>int</td>
<td>loopcount1</td>
<td>TIMES/END</td>
</tr>
<tr>
<td>int</td>
<td>loopcount2</td>
<td>TIMES2/END</td>
</tr>
<tr>
<td>int</td>
<td>loopcount3</td>
<td>TIMES3/END</td>
</tr>
<tr>
<td>int</td>
<td>loopcountinf</td>
<td>TIMESINFINITE</td>
</tr>
<tr>
<td>char</td>
<td>disk[256]</td>
<td>GETCURDATA</td>
</tr>
<tr>
<td>char</td>
<td>user[64]</td>
<td>GETCURDATA</td>
</tr>
<tr>
<td>char</td>
<td>type[16]</td>
<td>GETCURDATA</td>
</tr>
<tr>
<td>char</td>
<td>name[64]</td>
<td>GETCURDATA</td>
</tr>
<tr>
<td>int</td>
<td>expno</td>
<td>GETCURDATA, IEXPNO</td>
</tr>
<tr>
<td>int</td>
<td>procno</td>
<td>GETCURDATA, IPROCNO</td>
</tr>
<tr>
<td>char</td>
<td>disk2[256]</td>
<td>GETCURDATA2</td>
</tr>
<tr>
<td>char</td>
<td>user2[64]</td>
<td>GETCURDATA2</td>
</tr>
<tr>
<td>char</td>
<td>type2[16]</td>
<td>GETCURDATA2</td>
</tr>
<tr>
<td>char</td>
<td>name2[64]</td>
<td>GETCURDATA2</td>
</tr>
<tr>
<td>int</td>
<td>expno2</td>
<td>GETCURDATA2</td>
</tr>
<tr>
<td>int</td>
<td>procno2</td>
<td>GETCURDATA2</td>
</tr>
<tr>
<td>char</td>
<td>disk3[256]</td>
<td>GETCURDATA3</td>
</tr>
<tr>
<td>char</td>
<td>user3[64]</td>
<td>GETCURDATA3</td>
</tr>
<tr>
<td>char</td>
<td>type3[16]</td>
<td>GETCURDATA3</td>
</tr>
<tr>
<td>char</td>
<td>name3[64]</td>
<td>GETCURDATA3</td>
</tr>
<tr>
<td>int</td>
<td>expno3</td>
<td>GETCURDATA3</td>
</tr>
<tr>
<td>int</td>
<td>procno3</td>
<td>GETCURDATA3</td>
</tr>
<tr>
<td>char</td>
<td>namelist[10][64]</td>
<td>SETDATASET</td>
</tr>
<tr>
<td>char</td>
<td>dulist[10][256]</td>
<td>SETDATASET</td>
</tr>
<tr>
<td>char</td>
<td>userlist[10][64]</td>
<td>SETDATASET</td>
</tr>
</tbody>
</table>

Table 1.1
Table 1.1

1.13.3 Predefined general variables

The following list contains all predefined general variables, their types and
1.14 What to do after changing an AU program?

After changing a parameter in the userspecific AU program TopSpin must be updated to store the changed information in the AU program. This can be done with the command Show_meta (SM_SHOWP). This command also offers different arguments as following:

- **SM_RAW** ---- update raw data
- **SM_RAWP** ---- update acquisition parameters
- **SM_PROC** ---- update processed data
- **SM_PROCP** ---- update processing parameters
- **SM_ALL** ----- update data and parameters
- **SM_SHOWR** ---- switch to raw data

<table>
<thead>
<tr>
<th>type</th>
<th>variable</th>
<th>initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>i1</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>i2</td>
<td>0</td>
</tr>
<tr>
<td>int</td>
<td>i3</td>
<td>0</td>
</tr>
<tr>
<td>double</td>
<td>d1</td>
<td>0</td>
</tr>
<tr>
<td>double</td>
<td>d2</td>
<td>0</td>
</tr>
<tr>
<td>double</td>
<td>d3</td>
<td>0</td>
</tr>
<tr>
<td>float</td>
<td>f1</td>
<td>0</td>
</tr>
<tr>
<td>float</td>
<td>f2</td>
<td>0</td>
</tr>
<tr>
<td>float</td>
<td>f3</td>
<td>0</td>
</tr>
<tr>
<td>float</td>
<td>f998</td>
<td>0</td>
</tr>
<tr>
<td>float</td>
<td>f999</td>
<td>0</td>
</tr>
<tr>
<td>char</td>
<td>text[BUFSIZ/2]</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.2

initial values:
- **SM_SHOWP** ---- switch to processed data
- **SM_DEL** ---- removed data
- **SM_PEAK** ---- update peaks
- **SM_INT** ---- update integrals

Please note that changing the peaklist with a macro in the AU program does not require the command **SM_PEAK**. The changings are implemented automatically.
Chapter 2

Inventory of AU macros and Bruker library functions

2.1 Naming conventions

This chapter lists most AU macros and Bruker library functions that are available for AU programming. Simple macros with their short description are only mentioned in this chapter. More complex macros and AU functions are mentioned here and described more extensively in the following chapters. Table 2.1 explains the macro conventions used in this chapter.
Several AU macros that are described in this chapter require one or more arguments. These arguments can be constants or variables as described in Chapter 1.11.3. It is very important to use the correct type of argument in a macro call. The macros described in the tables of this chapter use the following arguments:

- integer : i1, i2, i3, eno, pno
- float : f1
- double : d1
- char-string: text, cmd, file, flag, mac, parm, parset, prog, shim, typ, dsk, usr, nam

Note that the arguments *i1, i2, i3, f1, d1* and *text* have the same names as the corresponding predefined general variables. The predefined general variables are easy to use because they do not need to be declared. You can, however, use your own variables as macro arguments.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXX</td>
<td>The macro can be typed &quot;as is&quot;. There is no further explanation for the macro in this manual.</td>
</tr>
<tr>
<td>XXX(arg1,arg2)</td>
<td>The macro XXX takes two arguments. Because the macro is easy to use, there is no further description in this manual.</td>
</tr>
<tr>
<td>XXX *</td>
<td>Like XXX, but there is a detailed description in one of the following chapters.</td>
</tr>
<tr>
<td>XXX(....) *</td>
<td>The macro XXX takes one or more arguments and its usage is described in one of the following chapters.</td>
</tr>
</tbody>
</table>

**Table 2.1 Macro conventions**
## 2.2 Macros for dataset handling

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETCURDATA *</td>
<td>Get the foreground dataset</td>
</tr>
<tr>
<td>SETCURDATA *</td>
<td>Make the current AU dataset available for subsequent AU statements</td>
</tr>
<tr>
<td>GETDATASET *</td>
<td>Prompt the user to specify a new dataset</td>
</tr>
<tr>
<td>DATASET(....) *</td>
<td>Set the current AU dataset</td>
</tr>
<tr>
<td>DATASET2(....) *</td>
<td>Set the 2nd dataset (like the TOPSPIN command <em>edc2</em>)</td>
</tr>
<tr>
<td>DATASET3(....) *</td>
<td>Set the 3rd dataset (like <em>edc2</em>)</td>
</tr>
<tr>
<td>GETCURDATA2</td>
<td>Read the 2nd dataset (like <em>edc2</em>)</td>
</tr>
<tr>
<td>GETCURDATA3</td>
<td>Read the 3rd dataset (like <em>edc2</em>)</td>
</tr>
<tr>
<td>DEXPNO *</td>
<td>Decrease the experiment number by one</td>
</tr>
<tr>
<td>IEXPNO *</td>
<td>Increase the experiment number by one</td>
</tr>
<tr>
<td>REXPNO(eno) *</td>
<td>Set the experiment number to the value of <em>eno</em></td>
</tr>
<tr>
<td>DPROCNO *</td>
<td>Decrease the processing number by one</td>
</tr>
<tr>
<td>IPROCNO *</td>
<td>Increase the processing number by one</td>
</tr>
<tr>
<td>RPROCNO(i1) *</td>
<td>Set the processing number to the value of <em>i1</em></td>
</tr>
<tr>
<td>GDATASETLIST</td>
<td>Prompt the user to enter a dataset list filename and read its contents</td>
</tr>
<tr>
<td>GLIST</td>
<td>Prompt the user to enter the dataset list filename and read its contents. In addition to the GDATASETLIST macro, GLIST also expects a pulse program and a parameter set name in the dataset list file.</td>
</tr>
<tr>
<td>DDATASETLIST</td>
<td>Decrement to the previous entry in the dataset list</td>
</tr>
<tr>
<td>IDATASETLIST</td>
<td>Increment to the next entry in the dataset list</td>
</tr>
<tr>
<td>RDATASETLIST(i1)</td>
<td>Read the dataset at position <em>i1</em> of the dataset list and make it the current AU dataset</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IFEODATASETLIST</td>
<td>Checks if the end of the dataset list is reached. The answer is true if there is no further entry.</td>
</tr>
<tr>
<td>SETDATASET</td>
<td>Set the current AU dataset to the one currently defined by the dataset list</td>
</tr>
<tr>
<td>DU(dsk)</td>
<td>Set the disk unit (top level data directory) to (dsk)</td>
</tr>
<tr>
<td>SETUSER(usr)</td>
<td>Set the user name to the user (usr)</td>
</tr>
<tr>
<td>RE(name)</td>
<td>Read the dataset name.</td>
</tr>
<tr>
<td>WRA(eno) *</td>
<td>Copy the raw data to the experiment number eno</td>
</tr>
<tr>
<td>WRP(pno) *</td>
<td>Copy the processed data to the processing number pno</td>
</tr>
<tr>
<td>WRPA(....) *</td>
<td>Copy the raw and processed data to the specified dataset</td>
</tr>
<tr>
<td>VIEWDATA *</td>
<td>Show the current AU program dataset in a new window or activate the window that contains this dataset.</td>
</tr>
<tr>
<td>VIEWDATA_SAMEWIN*</td>
<td>Show the current AU program dataset in the current window.</td>
</tr>
<tr>
<td>AUDITCOMMENTA (cmt)</td>
<td>Add a user comment to the acquisition audit trail (audita.txt)</td>
</tr>
<tr>
<td>AUDITCOMMENTP(cmt)</td>
<td>Add a user comment to the processing audit trail (auditp.txt)</td>
</tr>
<tr>
<td>GDCHECK</td>
<td>generate checksum, making the processing audit trail consistent</td>
</tr>
<tr>
<td>GDCHECKRAW</td>
<td>generate checksum, making the raw audit trail consistent</td>
</tr>
<tr>
<td>ACQUPATH(x)</td>
<td>Returns the path of the file (x) in the acquisition data directory (ACQU)</td>
</tr>
<tr>
<td>PROCPATH(x)</td>
<td>Returns the path of the file (x) in the processed data directory (PROCNO)</td>
</tr>
<tr>
<td>DELETEPROCCDATA</td>
<td>Delete processed data.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>DELETEIMAGINARY-DATA</td>
<td>Delete imaginary processed data.</td>
</tr>
<tr>
<td>DELETERAWDATA</td>
<td>Delete raw data.</td>
</tr>
<tr>
<td>DELETEPROCNO</td>
<td>Delete processed data directory (PROCNO).</td>
</tr>
<tr>
<td>DELETEEXPNO</td>
<td>Delete raw data directory (EXPNO).</td>
</tr>
<tr>
<td>DELETENAME</td>
<td>Delete data directory (NAME).</td>
</tr>
</tbody>
</table>
### 2.3 Macros prompting the user for input

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETDOUBLE(text,d1) *</td>
<td>Prompt the user to enter a double value</td>
</tr>
<tr>
<td>GETFLOAT(text,f1) *</td>
<td>Prompt the user to enter a float value</td>
</tr>
<tr>
<td>GETINT(text,i1) *</td>
<td>Prompt the user to enter an integer value</td>
</tr>
<tr>
<td>GETSTRING(text,nam) *</td>
<td>Prompt the user to enter a text string</td>
</tr>
</tbody>
</table>

### 2.4 Macros handling Topspin parameters

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETPROSOL *</td>
<td>Copy the probehead and solvent dependent parameters to the corresponding acquisition parameters</td>
</tr>
<tr>
<td>FETCHPAR(par,&amp;val) *</td>
<td>Get an acquisition or processing parameter</td>
</tr>
<tr>
<td>FETCHPAR1(par,&amp;val)</td>
<td>Get an F1 dimension parameter (2D acquisition/processing)</td>
</tr>
<tr>
<td>FETCHPAR3(par,&amp;val)</td>
<td>Get an F1 dimension parameter (3D acquisition/processing)</td>
</tr>
<tr>
<td>FETCHPARS(par,&amp;val) *</td>
<td>Get a status parameter (acquisition and processing)</td>
</tr>
<tr>
<td>FETCHPAR1S(par,&amp;val)</td>
<td>Get an F1 dimension status parameter (2D)</td>
</tr>
<tr>
<td>FETCHPAR3S(par,&amp;val)</td>
<td>Get an F1 dimension status parameter (3D)</td>
</tr>
<tr>
<td>FETCHPARN(dir,par,&amp;val)</td>
<td>Get a parameter from specified direction (nD)</td>
</tr>
<tr>
<td>FETCHPARNS(dir,par,&amp;val)</td>
<td>Get a status parameter from specified direction (nD)</td>
</tr>
<tr>
<td>STOREPAR(par,val) *</td>
<td>Store an acquisition, processing or output parameter</td>
</tr>
<tr>
<td>STOREPAR1(par,val)</td>
<td>Store an F1 dimension parameter (2D)</td>
</tr>
<tr>
<td>STOREPAR3(par,val)</td>
<td>Store an F1 dimension parameter (3D)</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>STOREPARS(par, val)</td>
<td>Store a status parameter (acquisition and processing)</td>
</tr>
<tr>
<td>STOREPAR1S(par, val)</td>
<td>Store an F1 dimension status parameter (2D)</td>
</tr>
<tr>
<td>STOREPAR3S(par, val)</td>
<td>Store an F1 dimension status parameter (3D)</td>
</tr>
<tr>
<td>STOREPARN(dir, par, &amp;val)</td>
<td>Store a parameter to specified direction (nD)</td>
</tr>
<tr>
<td>STOREPARNs(dir, par, &amp;val)</td>
<td>Store a status parameter to specified direction (nD)</td>
</tr>
<tr>
<td>FETCHPARN(par, &amp;val)</td>
<td>Get a tomography measurement parameter</td>
</tr>
<tr>
<td>STOREPARM(par, val)</td>
<td>Store a tomography measurement parameter</td>
</tr>
<tr>
<td>FETCHT1PAR(par, &amp;val)</td>
<td>Get a T1 parameter</td>
</tr>
<tr>
<td>STORET1PAR(par, val)</td>
<td>Store a T1 parameter</td>
</tr>
<tr>
<td>FETCHDOSYPAR(par, &amp;val)</td>
<td>Get a dosy (eddosy) parameter</td>
</tr>
<tr>
<td>STOREDOSYPAR(par, val)</td>
<td>Store a dosy (eddosy) parameter</td>
</tr>
<tr>
<td>RPAR(parset, typ) *</td>
<td>Read a parameter set to the current dataset</td>
</tr>
<tr>
<td>WPAR(parset, typ) *</td>
<td>Write the current dataset parameters to a parameter set</td>
</tr>
<tr>
<td>DELPAR(parset)</td>
<td>Delete the parameter set parset</td>
</tr>
</tbody>
</table>

*PARSET* is not used in any AU program


### 2.5 Acquisition macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZG</td>
<td>Start acquisition; if raw data already exist, they are overwritten</td>
</tr>
<tr>
<td>GO</td>
<td>Continue the acquisition on already existing raw data by adding to them</td>
</tr>
<tr>
<td>II</td>
<td>Initialize acquisition interface</td>
</tr>
<tr>
<td>RGA</td>
<td>Automatic receiver gain adjustment</td>
</tr>
<tr>
<td>MAKE_ZERO_FID</td>
<td>Create an empty FID</td>
</tr>
<tr>
<td>DEG90</td>
<td>Determine 90° pulse automatically</td>
</tr>
<tr>
<td>GPULPROGLIST</td>
<td>Prompt the user to enter the name of a pulse program list file and read its contents</td>
</tr>
<tr>
<td>DPULPROGLIST</td>
<td>Decrement to the previous name in the pulse program list</td>
</tr>
<tr>
<td>IPULPROGLIST</td>
<td>Increment to the next name in the pulse program list</td>
</tr>
<tr>
<td>RPULPROGLIST(i1)</td>
<td>Read the pulse program name in position i1 of the pulse program list and write it to the acquisition parameters</td>
</tr>
<tr>
<td>SETPULPROG</td>
<td>Store the current pulse program name from the pulse program list</td>
</tr>
<tr>
<td>IFEOPULPROGLIST</td>
<td>Check if the end of the pulse program list is reached. The answer is true if there is no further entry.</td>
</tr>
</tbody>
</table>

### 2.6 Macros handling the shim unit and the sample changer

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOGAIN</td>
<td>Optimize lock gain</td>
</tr>
<tr>
<td>AUTOPHASE</td>
<td>Optimize lock phase</td>
</tr>
<tr>
<td>AUTOSHIM_ON</td>
<td>Turn autoshim on</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AUTOSHIM_OFF</td>
<td>Turn autoshim off</td>
</tr>
<tr>
<td>EJ</td>
<td>Eject sample from the magnet</td>
</tr>
<tr>
<td>IJ</td>
<td>Insert sample into the magnet</td>
</tr>
<tr>
<td>LOCK_ON</td>
<td>Turn lock on</td>
</tr>
<tr>
<td>LOCK_OFF</td>
<td>Turn lock off</td>
</tr>
<tr>
<td>ROT</td>
<td>Turn rotation on (use value RO from acquisition parameters)</td>
</tr>
<tr>
<td>ROTOFF</td>
<td>Turn rotation off and wait until rotation was turned off</td>
</tr>
<tr>
<td>LOPO</td>
<td>Set the lock parameters (lock power, lock gain, loop filter, loop time and loop gain)</td>
</tr>
<tr>
<td>LFILTER(i1)</td>
<td>Set the loop filter to the value of i1</td>
</tr>
<tr>
<td>LG</td>
<td>Auto-adjust the lock gain</td>
</tr>
<tr>
<td>LGAIN(f1)</td>
<td>Set the loop gain to the value of f1</td>
</tr>
<tr>
<td>LO(f1)</td>
<td>Set the lock power to the value of f1</td>
</tr>
<tr>
<td>LTIME(f1)</td>
<td>Set the loop time to the value of f1</td>
</tr>
<tr>
<td>LOCK</td>
<td>Lock according to the parameters LOCNUC and SOLVENT using the lock parameters from the edlock table</td>
</tr>
<tr>
<td>RSH(file)</td>
<td>Read the shim values from the specified file</td>
</tr>
<tr>
<td>SETSH(shim,i1)</td>
<td>Set one shim to the value of i1</td>
</tr>
<tr>
<td>SWEEP_ON</td>
<td>Turn the lock-sweep on</td>
</tr>
<tr>
<td>SWEEP_OFF</td>
<td>Turn the lock-sweep off</td>
</tr>
<tr>
<td>WSH(file)</td>
<td>Write the shim values to the specified file</td>
</tr>
<tr>
<td>TUNE(file)</td>
<td>Start autoshimming with the specified tune file</td>
</tr>
<tr>
<td>TUNESX</td>
<td>Start autoshimming with the tune file defined by the currently defined probehead and solvent</td>
</tr>
</tbody>
</table>
## 2.7 Macros handling the temperature unit

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TESET</td>
<td>Set the temperature on the temperature unit to the value of the acquisition parameter TE.</td>
</tr>
<tr>
<td>TEGET</td>
<td>Get the temperature from the temperature unit and store it in the acquisition status parameter TE.</td>
</tr>
<tr>
<td>TE2SET</td>
<td>Set the temperature on the second regulator of the temperature unit to the value of the acquisition parameter TE2.</td>
</tr>
<tr>
<td>TE2GET</td>
<td>Get the temperature from the second regulator of the temperature unit and store it in the acquisition status parameter TE2.</td>
</tr>
<tr>
<td>TEREADY(i1,f1)</td>
<td>After the temperature is set, wait until it is accurate to f1 degrees for at least 10 sec., then wait i1 seconds for stabilization.</td>
</tr>
<tr>
<td>TE2READY(i1,f1)</td>
<td>After the second temperature is set, wait until it is accurate to f1 degrees for at least 10 sec., then wait i1 seconds for stabilization.</td>
</tr>
<tr>
<td>TEPAR(file)</td>
<td>Read a file with parameter settings for the temperature unit.</td>
</tr>
<tr>
<td>GVTLIST</td>
<td>Prompt the user to enter the variable temperature list name and read its contents.</td>
</tr>
<tr>
<td>RVTLIST</td>
<td>Read the contents of the variable temperature list file defined by the acquisition parameter VTLIST.</td>
</tr>
<tr>
<td>DVTLIST</td>
<td>Decrement to the previous value in the vtlist.</td>
</tr>
<tr>
<td>IVTLIST</td>
<td>Increment to the next value in the vtlist.</td>
</tr>
<tr>
<td>VT</td>
<td>Read and set the temperature according to the current value of the vtlist.</td>
</tr>
</tbody>
</table>
2.8 Macros handling the MAS unit

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASE</td>
<td>Eject sample from MAS unit</td>
</tr>
<tr>
<td>MASI</td>
<td>Insert sample into MAS unit</td>
</tr>
<tr>
<td>MASR</td>
<td>Set spinning rate according to the acquisition parameter MASR</td>
</tr>
<tr>
<td>MASRGET</td>
<td>Get spinning rate from the MAS unit and store it in the status acquisition parameters</td>
</tr>
<tr>
<td>MASG(i1)</td>
<td>Start spinning of sample in MAS with at the most i1 retries</td>
</tr>
<tr>
<td>MASH</td>
<td>Halt spinning of sample in MAS</td>
</tr>
</tbody>
</table>

2.9 1D processing macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Automatic baseline correction (creates intrng file !)</td>
</tr>
<tr>
<td>ABSD</td>
<td>Automatic baseline correction with DISNMR algorithm (creates intrng file !)</td>
</tr>
<tr>
<td>ABSF</td>
<td>Automatic baseline correction between limits ABSF1 and ABSF2</td>
</tr>
<tr>
<td>APK</td>
<td>Automatic phase correction</td>
</tr>
<tr>
<td>APK0</td>
<td>Zero order automatic phase correction</td>
</tr>
<tr>
<td>APK1</td>
<td>First order automatic phase correction</td>
</tr>
<tr>
<td>APKF</td>
<td>Automatic phase correction using the spectral region determined by ABSF2 and ABSF1 for the calculation of the phase values.</td>
</tr>
<tr>
<td>APK0F</td>
<td>Zero order automatic phase correction using the spectral region determined by ABSF2 and ABSF1 for the calculation of the phase values.</td>
</tr>
<tr>
<td>APKS</td>
<td>Automatic phase correction especially suitable for polymer spectra</td>
</tr>
<tr>
<td>BC</td>
<td>Baseline correction of FID (DC correction)</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------------------</td>
</tr>
<tr>
<td>BCM</td>
<td>User defined spectrum baseline correction</td>
</tr>
<tr>
<td>CONVDTA(eno)</td>
<td>Convert digitally filtered FID into analogue (conventional) form</td>
</tr>
<tr>
<td>EF</td>
<td>Exponential window multiplication + Fourier transform</td>
</tr>
<tr>
<td>EFP</td>
<td>Exponential window multiplication + Fourier transform + phase correction using the processing parameters PHC0 and PHC1</td>
</tr>
<tr>
<td>EM</td>
<td>Exponential window multiplication of FID</td>
</tr>
<tr>
<td>FMC</td>
<td>Fourier Transform + magnitude calculation</td>
</tr>
<tr>
<td>FP</td>
<td>Fourier Transform + phase correction using the processing parameters PHC0 and PHC1</td>
</tr>
<tr>
<td>FT</td>
<td>Fourier Transform</td>
</tr>
<tr>
<td>GENFID(eno)</td>
<td>Create FID from processed data</td>
</tr>
<tr>
<td>GF</td>
<td>Gaussian window multiplication + Fourier Transform</td>
</tr>
<tr>
<td>GFP</td>
<td>Gaussian window multiplication + Fourier Transform + phase correction using the processing parameters PHC0 and PHC1</td>
</tr>
<tr>
<td>GM</td>
<td>Gaussian window multiplication</td>
</tr>
<tr>
<td>HT</td>
<td>Hilbert Transform</td>
</tr>
<tr>
<td>IFT</td>
<td>Inverse Fourier Transform</td>
</tr>
<tr>
<td>MC</td>
<td>Magnitude calculation</td>
</tr>
<tr>
<td>PK</td>
<td>Phase correction using the processing parameters PHC0 and PHC1</td>
</tr>
<tr>
<td>PS</td>
<td>Power spectrum calculation</td>
</tr>
<tr>
<td>QSIN</td>
<td>Squared sine window multiplication</td>
</tr>
<tr>
<td>SAB</td>
<td>Spline baseline correction using base_info file</td>
</tr>
<tr>
<td>SINM</td>
<td>Sine window multiplication</td>
</tr>
<tr>
<td>SINO</td>
<td>Calculate signal to noise ratio</td>
</tr>
<tr>
<td>SREF</td>
<td>Automatic spectral referencing using 2Hlock parameters</td>
</tr>
<tr>
<td>TM</td>
<td>Trapezoidal window multiplication</td>
</tr>
</tbody>
</table>
Note that 1D processing macros which access raw data, execute the corresponding command with the option `same`. For example, FT executes the command `ft same`.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRF</td>
<td>Processing of the raw data according to the currently defined processing parameters</td>
</tr>
<tr>
<td>TRFP</td>
<td>Processing of the processed data according to the currently defined processing parameters</td>
</tr>
<tr>
<td>UWM</td>
<td>User-defined window multiplication</td>
</tr>
</tbody>
</table>
2.10 Peak picking, integration and miscellaneous macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP</td>
<td>Peak picking according to currently set processing parameters</td>
</tr>
<tr>
<td>PPH</td>
<td>Like PP, but with a peak histogram along the listing</td>
</tr>
<tr>
<td>PPP</td>
<td>Like PP, but the output is written to the file peaklist in the current processing data directory (PROCNO)</td>
</tr>
<tr>
<td>PPJ</td>
<td>Like PP, but store peaks in JCAMP-DX format</td>
</tr>
<tr>
<td>LI</td>
<td>List integrals according to the currently defined intrng file. The macro ABS can be used to create an intrng file.</td>
</tr>
<tr>
<td>LIPP</td>
<td>List integrals and all peaks in the integral ranges</td>
</tr>
<tr>
<td>LIPPF</td>
<td>Like LIPP, but works always on the full spectrum</td>
</tr>
<tr>
<td>PP2D</td>
<td>Perform peak picking on a 2D dataset.</td>
</tr>
<tr>
<td>RMISC(typ,file)</td>
<td>Read a file from one of the following list types: base_info, baslpts, intrng, peaklist or reg</td>
</tr>
<tr>
<td>WMISC(typ,file)</td>
<td>Write a base_info, baslpts, intrng, peaklist or reg file to its lists directory</td>
</tr>
</tbody>
</table>

2.11 Macros for algebraic operations on datasets

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Add 2nd and 3rd dataset and put the result into the current dataset. The 3rd dataset is multiplied by DC.</td>
</tr>
<tr>
<td>ADDFID</td>
<td>Add two FIDs multiplying one of them with DC.</td>
</tr>
<tr>
<td>ADDC</td>
<td>Add the constant DC to the current dataset</td>
</tr>
<tr>
<td>AND</td>
<td>Put logical &quot;and&quot; of 2nd and 3rd dataset into the current dataset</td>
</tr>
<tr>
<td>DIV</td>
<td>Divide 2nd and 3rd dataset and put the result into the current dataset. The 3rd dataset is multiplied by DC.</td>
</tr>
<tr>
<td>DT</td>
<td>Calculate the first derivative of the dataset</td>
</tr>
<tr>
<td>FILT</td>
<td>Apply a software digital filter to the current dataset</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>LS</td>
<td>Left shift spectrum or FID by NSP points</td>
</tr>
<tr>
<td>MUL</td>
<td>Multiply 2nd and 3rd dataset and put the result into the current dataset. The 3rd dataset is multiplied by DC.</td>
</tr>
<tr>
<td>MULC</td>
<td>Multiply the current dataset with DC</td>
</tr>
<tr>
<td>NM</td>
<td>Negate current spectrum</td>
</tr>
<tr>
<td>RS</td>
<td>Right shift spectrum or FID by NSP points</td>
</tr>
<tr>
<td>RV</td>
<td>Reverse the spectrum</td>
</tr>
<tr>
<td>ZF</td>
<td>Zero the spectrum (1r,1i)</td>
</tr>
<tr>
<td>ZP</td>
<td>Zero the first NZP points of the spectrum or FID</td>
</tr>
</tbody>
</table>
### 2.12 Deconvolution macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDCON</td>
<td>Gaussian deconvolution of the peaks automatically picked according to the currently set processing parameters</td>
</tr>
<tr>
<td>LDCON</td>
<td>Lorentzian deconvolution of the peaks automatically picked according to the currently set processing parameters</td>
</tr>
<tr>
<td>MDCON</td>
<td>Mixed Gaussian/Lorentzian deconvolution of the peaks in the peaklist file. The peaklist file can be created with the <em>ppp</em> command and it can be modified using the <em>edmisc</em> command.</td>
</tr>
</tbody>
</table>

### 2.13 2D processing macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS1</td>
<td>Baseline correction in F1 dimension</td>
</tr>
<tr>
<td>ABS2</td>
<td>Baseline correction in F2 dimension</td>
</tr>
<tr>
<td>ABSD1</td>
<td>Baseline correction in F1 dimension using the DISNMR algorithm</td>
</tr>
<tr>
<td>ABSD2</td>
<td>Baseline correction in F2 dimension using the DISNMR algorithm</td>
</tr>
<tr>
<td>ABSOT1</td>
<td>Trapezoidal baseline correction in F1 dimension using a slightly different algorithm than abst1</td>
</tr>
<tr>
<td>ABSOT2</td>
<td>Trapezoidal baseline correction in F2 dimension using a slightly different algorithm than abst2</td>
</tr>
<tr>
<td>ABST1</td>
<td>Trapezoidal baseline correction in F1 dimension using the processing parameters ABSF1, ABSF2, SIGF1, SIGF2</td>
</tr>
<tr>
<td>ABST2</td>
<td>Trapezoidal baseline correction in F2 dimension using the processing parameters ABSF1, ABSF2, SIGF1, SIGF2</td>
</tr>
<tr>
<td>ADD2D</td>
<td>Add the processed data of the 2nd dataset to the current dataset.</td>
</tr>
<tr>
<td>ADDSER</td>
<td>Add the raw data of the 2nd dataset to the current dataset.</td>
</tr>
<tr>
<td>BCM1</td>
<td>Baseline correction of all columns using the coefficients that were obtained with a manual 1D baseline correction</td>
</tr>
<tr>
<td>BCM2</td>
<td>Baseline correction of all rows using the coefficients that were obtained with a manual 1D baseline correction</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>INVSF</td>
<td>Interchange the frequencies of the two dimensions</td>
</tr>
<tr>
<td>LEVCALC</td>
<td>Calculate the levels for the contour representation of the 2D matrix</td>
</tr>
<tr>
<td>PTILT</td>
<td>Tilt the 2D matrix by an arbitrary angle</td>
</tr>
<tr>
<td>PTILT1</td>
<td>Tilt the 2D matrix along its central vertical line</td>
</tr>
<tr>
<td>REV1</td>
<td>Reverse the spectrum in F1 dimension</td>
</tr>
<tr>
<td>REV2</td>
<td>Reverse the spectrum in F2 dimension</td>
</tr>
<tr>
<td>SUB1</td>
<td>Subtract 1D spectrum in F1 dimension (no change in sign !)</td>
</tr>
<tr>
<td>SUB2</td>
<td>Subtract 1D spectrum in F2 dimension (no change in sign !)</td>
</tr>
<tr>
<td>SUB1D1</td>
<td>Subtract 1D spectrum in F1 dimension</td>
</tr>
<tr>
<td>SUB1D2</td>
<td>Subtract 1D spectrum in F2 dimension</td>
</tr>
<tr>
<td>SYM</td>
<td>Symmetrize COSY spectrum</td>
</tr>
<tr>
<td>SYMA</td>
<td>Symmetrize phase sensitive COSY spectrum</td>
</tr>
<tr>
<td>SYMJ</td>
<td>Symmetrize J-resolved spectrum</td>
</tr>
<tr>
<td>TILT</td>
<td>Tilt J-resolved spectrum by an internally calculated angle</td>
</tr>
<tr>
<td>XF1</td>
<td>Fourier transform in F1 dimension</td>
</tr>
<tr>
<td>XF1P</td>
<td>Phase correction in F1 dimension using the processing parameters PHC0 and PHC1</td>
</tr>
<tr>
<td>XF2</td>
<td>Fourier transform in F2 dimension</td>
</tr>
<tr>
<td>XF2P</td>
<td>Phase correction in F2 dimension using the processing parameters PHC0 and PHC1</td>
</tr>
<tr>
<td>XFB</td>
<td>Fourier transform in both dimensions</td>
</tr>
<tr>
<td>XFBP</td>
<td>Phase correction in both dimensions</td>
</tr>
<tr>
<td>XF1M</td>
<td>Magnitude calculation in F1 dimension</td>
</tr>
<tr>
<td>XF2M</td>
<td>Magnitude calculation in F2 dimension</td>
</tr>
<tr>
<td>XFBM</td>
<td>Magnitude calculation in both dimensions</td>
</tr>
<tr>
<td>XF1PS</td>
<td>Power spectrum in F1 dimension</td>
</tr>
<tr>
<td>XF2PS</td>
<td>Power spectrum in F2 dimension</td>
</tr>
<tr>
<td>XFBPS</td>
<td>Power spectrum in both dimensions</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>XHT1</td>
<td>Hilbert Transform in F1 dimension</td>
</tr>
<tr>
<td>XHT2</td>
<td>Hilbert transform in F2 dimension</td>
</tr>
<tr>
<td>XIF1</td>
<td>Inverse Fourier transform in F1 dimension</td>
</tr>
<tr>
<td>XIF2</td>
<td>Inverse Fourier transform in F2 dimension</td>
</tr>
<tr>
<td>XTRF</td>
<td>2D processing according to processing parameter flags (starts always on the raw data !)</td>
</tr>
<tr>
<td>XTRF2</td>
<td>2D processing according to F2 processing parameter flags only (starts always on the raw data !)</td>
</tr>
<tr>
<td>XTRFP</td>
<td>2D Processing according to the processing parameter flags</td>
</tr>
<tr>
<td>XTRFP1</td>
<td>2D processing according to the F1 processing parameter flags only</td>
</tr>
<tr>
<td>XTRFP2</td>
<td>2D processing according to the F2 processing parameter flags only</td>
</tr>
<tr>
<td>ZERT1</td>
<td>Zero a region of each column (F1). The region is determined by ABSF1/ABSF2 (first column) and SIGF1/SIGF2 (last column)</td>
</tr>
<tr>
<td>ZERT2</td>
<td>Zero a region of each row (F1). The region is determined by ABSF1/ABSF2 (first row) and SIGF1/SIGF2 (last row)</td>
</tr>
<tr>
<td>GENSE(n</td>
<td>Create a 2D series file from the processed data</td>
</tr>
<tr>
<td>o)</td>
<td></td>
</tr>
</tbody>
</table>

Note that 2D processing macros which access raw data, execute the corresponding command with the option **same**. For example, XFB executes the command **xfb same**.
## 2.14 Macros reading and writing projections etc.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1SUM(i1,i2,pno)</td>
<td>Read sum of columns from i1 to i2 into the 1D processing number pno</td>
</tr>
<tr>
<td>F2SUM(i1,i2,pno)</td>
<td>Read sum of rows from i1 to i2 into the 1D processing number pno</td>
</tr>
<tr>
<td>F1DISCO(i1,i2,i3,pno)</td>
<td>Read disco projection between i1 and i2 columns with reference row i3 into the 1D processing number pno</td>
</tr>
<tr>
<td>F2DISCO(i1,i2,i3,pno)</td>
<td>Read disco projection between i1 and i2 rows with reference column i3 into the 1D processing number pno</td>
</tr>
<tr>
<td>F1PROJN(i1,i2,pno)</td>
<td>Read partial negative projection between columns i1 and i2 into the 1D processing number pno</td>
</tr>
<tr>
<td>F1PROJP(i1,i2,pno)</td>
<td>Read partial positive projection between columns i1 and i2 into the 1D processing number pno</td>
</tr>
<tr>
<td>F2PROJN(i1,i2,pno)</td>
<td>Read partial negative projection between rows i1 and i2 into the 1D processing number pno</td>
</tr>
<tr>
<td>F2PROJP(i1,i2,pno)</td>
<td>Read partial positive projection between rows i1 and i2 into the 1D processing number pno</td>
</tr>
<tr>
<td>RHNP(pno)</td>
<td>Read horizontal (F2) negative projection into the 1D processing number pno</td>
</tr>
<tr>
<td>RHPP(pno)</td>
<td>Read horizontal (F2) positive projection into the 1D processing number pno</td>
</tr>
<tr>
<td>RSC(i1,pno) *</td>
<td>Read column i1 of 2D into the 1D processing number pno</td>
</tr>
<tr>
<td>RSR(i1,pno) *</td>
<td>Read row i1 of 2D into the 1D processing number pno</td>
</tr>
</tbody>
</table>
### 2.15 3D processing macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVNP(pno) *</td>
<td>Read vertical (F1) negative projection into the 1D processing number pno.</td>
</tr>
<tr>
<td>RVPP(pno) *</td>
<td>Read vertical (F1) positive projection into the 1D processing number pno.</td>
</tr>
<tr>
<td>RSER(i1,eno,pno) *</td>
<td>Read row i1 of 2D raw data into the eno and pno.</td>
</tr>
<tr>
<td>RSER2D(direc, i1,eno) *</td>
<td>Read plane number i1 in direction direc of 3D raw data into the eno.</td>
</tr>
<tr>
<td>WSC(i1,pno,eno,nam,usr,dsk) *</td>
<td>Write a column back into position i1 of a 2D dataset defined by pno, eno, nam, usr and dsk.</td>
</tr>
<tr>
<td>WSR(i1,pno,eno,nam,usr,dsk) *</td>
<td>Write a row back into position i1 of a 2D dataset defined by pno, eno, nam, usr and dsk.</td>
</tr>
<tr>
<td>WSER(i1,nam,eno,pno,dsk,usr)*</td>
<td>Write an FID back into position i1 of a 2D raw data defined by eno, pno, nam, dsk and usr.</td>
</tr>
<tr>
<td>WSERP(i1,nam,eno,pno,dsk,usr)</td>
<td>Write a processed FID back into position i1 of a 2D raw data defined by eno, pno, nam, dsk and usr.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF3(flag,dsk)</td>
<td>Fourier transform in F3 dimension. The flag can be &quot;y&quot; or &quot;n&quot; and determines whether the imaginary parts are stored or not. The processed are stored on disk unit dsk.</td>
</tr>
<tr>
<td>TF2(flag)</td>
<td>Fourier transform in F2 dimension (flag as in TF3).</td>
</tr>
<tr>
<td>TF1(flag)</td>
<td>Fourier transform in F1 dimension (flag as in TF3).</td>
</tr>
<tr>
<td>TF3P(flag)</td>
<td>Phase correction in F3 dimension (flag as in TF3).</td>
</tr>
<tr>
<td>TF2P(flag)</td>
<td>Phase correction in F2 dimension (flag as in TF3).</td>
</tr>
<tr>
<td>TF1P(flag)</td>
<td>Phase correction in F1 dimension (flag as in TF3).</td>
</tr>
<tr>
<td>TABS3</td>
<td>Automatic baseline correction in F3 dimension.</td>
</tr>
</tbody>
</table>
### 2.16 Spectral Width calculation macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETLIM</td>
<td>Get frequency of leftmost and rightmost peak from a 1D spectrum and adjust the sweep width of the 1D spectrum to the difference + 10%</td>
</tr>
<tr>
<td>GETLCOSY</td>
<td>Get frequency of leftmost and rightmost peak from a 1D spectrum and adjust the sweep width of a COSY spectrum to the difference + 10%</td>
</tr>
<tr>
<td>GETLXHCO</td>
<td>Get frequency of leftmost and rightmost peak from two 1D spectra and adjust the sweep width of an X-H correlation spectrum to the difference + 10%</td>
</tr>
<tr>
<td>GETLJRES</td>
<td>Get frequency of leftmost and rightmost peak from a 1D spectrum and adjust the sweep width of a J-RESolved spectrum to the difference + 10%</td>
</tr>
<tr>
<td>GETLINV</td>
<td>Get frequency of leftmost and rightmost peak from a 1D spectrum and adjust the sweep width of an INVerse spectrum to the difference + 10%</td>
</tr>
</tbody>
</table>

### 2.17 Plot editor related macros

- **XWP_LP** * Create a parameter listing for a plot with the plot editor.
- **XWP_PP** * Create a peak picking listing for a plot with the plot editor.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AUTOPLOT</strong> *</td>
<td>Plot the current dataset according to the plot editor layout defined by the processing parameter LAYOUT.</td>
</tr>
<tr>
<td><strong>AUTOPLOT_TO_FILE(filename)</strong> *</td>
<td>as AUTOPLOT except that the plot is not sent to the printer but store in the postscript file filename.</td>
</tr>
<tr>
<td><strong>DECLARE_PORTFOLIO</strong> *</td>
<td>Initialize the usage of other TOPSPIN portfolio AU macros. Required in XWIN-NMR and TOPSPIN ≤ 1.2. Obsolete in TOPSPIN ≥ 1.3.</td>
</tr>
<tr>
<td><strong>CREATE_PORTFOLIO(filename)</strong> *</td>
<td>Create the TOPSPIN portfolio filename.</td>
</tr>
<tr>
<td><strong>ADD_TO_PORTFOLIO(disk, user, name, expno, procno)</strong> *</td>
<td>Add the dataset that is specified with the arguments to the portfolio created with CREATE_PORTFOLIO.</td>
</tr>
<tr>
<td><strong>ADD_CURDAT_TO_PORTFOLIO</strong> *</td>
<td>Add the current dataset to the portfolio created with CREATE_PORTFOLIO.</td>
</tr>
<tr>
<td><strong>CLOSE_PORTFOLIO</strong> *</td>
<td>Close the definition for the portfolio created with CREATE_PORTFOLIO. Must be used before AUTOPLOT_* macros.</td>
</tr>
<tr>
<td><strong>AUTOPLOT_WITH_PORTFOLIO</strong> *</td>
<td>Plot the dataset(s) defined in the portfolio created with CREATE_PORTFOLIO according to the layout defined by the parameter LAYOUT.</td>
</tr>
<tr>
<td><strong>AUTOPLOT_WITH_PORTFOLIO_TO_FILE(filename)</strong> *</td>
<td>as AUTOPLOT_WITH_PORTFOLIO except that the plot is not sent to the printer but store in the postscript file filename.</td>
</tr>
<tr>
<td><strong>LAYOUT_ADD</strong></td>
<td>Add object to current layout</td>
</tr>
<tr>
<td><strong>LAYOUT_ADD_1D_OBJECT</strong></td>
<td>Add 1D object to current layout</td>
</tr>
<tr>
<td><strong>LAYOUT_ADD_PARAMETERS</strong></td>
<td>Add parameter object to current layout</td>
</tr>
<tr>
<td><strong>LAYOUT_BEGIN_FILE</strong></td>
<td>Open layout file</td>
</tr>
<tr>
<td><strong>LAYOUT_BEGIN_FILE</strong></td>
<td>Close layout file</td>
</tr>
</tbody>
</table>
2.18 Macros converting datasets

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROMJDX(....)</td>
<td>Convert a JCAMP-DX file to TOPSPIN data format</td>
</tr>
<tr>
<td>TOJDX(....)</td>
<td>Convert a dataset to JCAMP-DX 6.0 format</td>
</tr>
<tr>
<td>TOJDX5(....)</td>
<td>Convert a dataset to JCAMP-DX 5.0 format</td>
</tr>
<tr>
<td>JCONV(....)</td>
<td>Convert a Jeol dataset to Bruker TOPSPIN format</td>
</tr>
<tr>
<td>VCONV(....)</td>
<td>Convert a Varian dataset to Bruker TOPSPIN format</td>
</tr>
</tbody>
</table>

2.19 Macros to execute other AU programs, Topspin macros or commands

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPR_exec(....)</td>
<td>C-function for executing special TOPSPIN commands</td>
</tr>
<tr>
<td>WAIT_UNTIL(....)</td>
<td>Hold the AU program until the specified date and time</td>
</tr>
<tr>
<td>XAUA</td>
<td>Execute the acquisition AU program stored in AUNM (eda). The next line in the AU program is executed after the AU program AUNM has finished.</td>
</tr>
<tr>
<td>XAUP</td>
<td>Execute the processing AU program stored in AUNMP (edp). The next line in the AU program is immediately executed after the AU program AUNMP has been started.</td>
</tr>
</tbody>
</table>
2.20 Bruker library functions

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAUPW</td>
<td>Execute the processing AU program stored in AUNMP ((\text{edp})). Like XAUP, but now the next line in the AU program is executed after the AU program AUNMP has finished.</td>
</tr>
<tr>
<td>XAU(prog, arg)</td>
<td>Execute the AU program prog with the wait option.</td>
</tr>
<tr>
<td>XCMD(cmd)</td>
<td>Execute the TOPSPIN command for which no dedicated macro exists.</td>
</tr>
<tr>
<td>XMAC(mac)</td>
<td>Execute a TOPSPIN macro mac.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalcExpTime() *</td>
<td>Calculate the experiment time for the current experiment</td>
</tr>
<tr>
<td>PrintExpTime(....) *</td>
<td>Print the experiment time for the current experiment</td>
</tr>
<tr>
<td>check_pwd(usr) *</td>
<td>Prompt the user usr to enter a password</td>
</tr>
<tr>
<td>GetNmrSuperUser() *</td>
<td>Get the name of the current TOPSPIN superuser</td>
</tr>
<tr>
<td>getdir(....) *</td>
<td>Get all file names and/or directory names within a directory</td>
</tr>
<tr>
<td>freedir(....) *</td>
<td>Free memory allocated by getdir</td>
</tr>
<tr>
<td>dircp(....) *</td>
<td>Copy a file</td>
</tr>
<tr>
<td>dircp_err(i1) *</td>
<td>Return the error message that corresponds to the error return value of a dircp function call</td>
</tr>
<tr>
<td>fetchstorpl(....) *</td>
<td>Read or store one or several plot parameters</td>
</tr>
<tr>
<td>FileSelect(....) *</td>
<td>Display a list of files and allow to select a file</td>
</tr>
<tr>
<td>gethighest(....) *</td>
<td>Return the next highest unused experiment number of a dataset</td>
</tr>
<tr>
<td>getParamDirs(...)</td>
<td>List all directories specified for key.</td>
</tr>
<tr>
<td>getParfileDirForRead(...) *</td>
<td>Determines pathname of list file to be read.</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>getParfileDirForWrite(..)</td>
<td>Determines pathname of list file to be written.</td>
</tr>
<tr>
<td>getstan(....)</td>
<td>Return the pathname to the user’s current experiment directory.</td>
</tr>
<tr>
<td>getxwinvers(....)</td>
<td>Return the current version and patchlevel of TOPSPIN</td>
</tr>
<tr>
<td>mkudir(....)</td>
<td>Create a complete directory path</td>
</tr>
<tr>
<td>PathXWinNMR()</td>
<td>A class of functions which return pathnames to certain TOPSPIN directories</td>
</tr>
<tr>
<td>pow_next(i1)</td>
<td>Round i1 to the next larger power of two</td>
</tr>
<tr>
<td>Proc_err(....)</td>
<td>Show a message in a TOPSPIN dialog window</td>
</tr>
<tr>
<td>Show_status(text)</td>
<td>Show a string in the status line of TOPSPIN</td>
</tr>
<tr>
<td>showfile(file)</td>
<td>Show the contents of a file in a TOPSPIN window</td>
</tr>
<tr>
<td>ssleep(i1)</td>
<td>Pause in an AU program for i1 seconds</td>
</tr>
<tr>
<td>unlinkpr(....)</td>
<td>Delete all processed data files (1r, 1i, 2rr, 2ii etc.) of a dataset</td>
</tr>
</tbody>
</table>
### 2.21 Macros for loop control

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMES(n)</td>
<td>Execute the statements in the loop n times.</td>
</tr>
<tr>
<td>TIMES2(n)</td>
<td>Execute the statements in the loop n times. Normally used for the second level of nested loops.</td>
</tr>
<tr>
<td>TIMES3(n)</td>
<td>Execute the statements in the loop n times. Normally used for the second level of nested loops.</td>
</tr>
<tr>
<td>END</td>
<td>End of a loop.</td>
</tr>
<tr>
<td>STOP</td>
<td>Stop the AU program with the return value of AUERR.</td>
</tr>
<tr>
<td>STOPMSG(&quot;text&quot;)</td>
<td>Stop the AU program with the return value of AUERR and display the message &quot;text&quot;</td>
</tr>
</tbody>
</table>

### 2.22 Macros to return from an AU program

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORT</td>
<td>Abort the AU program or any of its subroutines with the return value of -1</td>
</tr>
<tr>
<td>ERRORABORT</td>
<td>Return from an AU program or any of its subroutines with the value of AUERR if it is less than 0</td>
</tr>
<tr>
<td>QUIT</td>
<td>Return from an AU program with the value of AUERR. QUIT is usually the last statement of the AU program code.</td>
</tr>
<tr>
<td>QUITMSG(text)</td>
<td>Print the text message and then return from the AU program with the value of AUERR. This is an alternative to QUIT.</td>
</tr>
<tr>
<td>STOP</td>
<td>Stop the AU program with the return value of AUERR.</td>
</tr>
<tr>
<td>STOPMSG(&quot;text&quot;)</td>
<td>Stop the AU program with the return value of AUERR and display the message &quot;text&quot;</td>
</tr>
</tbody>
</table>
Chapter 3

General AU macros

This chapter contains a description of all general AU macros which can be used for various purposes.
CPR_exec

NAME
CPR_exec - generic function for executing TOPSPIN commands

SYNTAX
CPR_exec(const char *command, int mode);

DESCRIPTION
CPR_exec is a generic function which can be used for executing TOPSPIN commands in AU. The first argument of CPR_exec is a string containing a Topspin command. The second argument must be one of the following values:

WAIT_TERM - wait for the command to finish, then start the next command
WAIT_START - wait for the command to start, then start the next command
CONT_EX - start the command and immediately start the next command

Practically all dedicated macros which execute a TOPSPIN command call CPR_exec with WAIT_TERM. For example, the macro FT is defined as

FT {SETCURDATA AUERR=CPR_exec( "ft same",WAIT_TERM);}

The CPR_exec return value allows you to check for successfully execution. The return value of CPR_Exec is NORM_TERM (=0) for normal termination or ERR_TERM (=1) for error termination.

WAIT_START or CONT_EX can be used if asynchronous execution is required. For example, the AU macro XAUP uses WAIT_START to allow data simultaneous processing and acquisition in automation. Note that using WAIT_START and CONT_EX does not allow you to check the return value for successful exection.

For most commands a dedicated AU macro is available, like ZG for zg and FT for ft. If you want to use TOPSPIN commands for which no dedicated macro exist, e.g. editor commands or commands with special arguments, then you can use the generic macro XCMD which takes only one
argument, the TOPSPIN command and is started with WAIT_TERM. XCMD is defined as:

\[
\text{XCMD}(\text{cmd}) \{ \text{SETCURDATA AU-} \\
\text{ERR=CPR\_exec(cmd,WAIT\_TERM);} \}
\]

In fact, the only reason to use CPR\_exec explicitly is to start a command with WAIT\_START or CONT\_EX, i.e. to run commands simultaneously.

Note that dedicated macros and XCMD call SETCURDATA before they do their actual task. This ensures that they operate on the current AU dataset. If you use CPR\_exec explicitly, it is recommended to precede it with SETCURDATA. Note that in the example below, CPR\_exec is preceded by the macro ZG which implicitly calls SETCURDATA.

In summary:

- Use dedicated AU macros whenever you can
- Use XCMD when no dedicated macro is available
- Use CPR\_EXEC when you want to use WAIT\_START or CONT\_EX

CPR\_exec is part of the uni library which is delivered with TOPSPIN.

EXAMPLE

The following AU program gets the foreground dataset, runs an acquisition, starts the Fourier Transform and, after this has started, continues an acquisition on the next experiment number:

\[
\text{TIMES}(10) \\
\text{ZG} \\
\text{CPR\_exec("ft", WAIT\_START);} \\
\text{IEXP\_NO} \\
\text{END} \\
\text{QUIT}
\]

SEE ALSO

XCMD - generic macro to execute commands for which no dedicated macro exists
SETCURDATA - make the current AU dataset available for subsequent AU statements
XAU

NAME

XAU - generic function for executing AU programs

SYNTAX

XAU(prog, arg)

DESCRIPTION

XAU is a general macro to execute (and, if necessary compile) AU programs. The macro takes two arguments:

prog - the AU program to be executed

arg - arguments

The second argument can be:

• any character string in """" containing one or more arguments
• cmd
to transfer all arguments from calling AU program
• ""
  no arguments are propagated

In TOPSPIN 2.0 and older, XAU requires only one argument, the AU program to be executed, automatically propagates all arguments. In TOPSPIN 2.1 and newer, you can freely choose the arguments to be transferred.

User defined AU programs containing XAU macros must be modified. You could simply extend an XAU call with the extra argument cmd. As such, your AU program behaves exactly the same as in your previous version, namely propagating all caller arguments. The reason, however, that XAU was modified is that you normally do not want to propagate all arguments. In this case, you can replace XAU by XCMD, in which case the AU program can still be used with TOPSPIN 2.0 and older. In that case you can:

Replace XAU("auprog") by XCMD("auprog") or XAU("auprog", "")
to propagate no arguments
Replace XAU(auprog) by XAU("auprog", cmd)
to propagate all arguments
Use XCMD("auprog arg1 arg2") or XAU("auprog", "arg1 arg2")
to specify new arguments arg1 and arg2.

SEE ALSO
XCMD - generic function for executing TOPSPIN commands
XCMD

NAME

XCMD - generic function for executing TOPSPIN commands

SYNTAX

XCMD(char *command)

DESCRIPTION

XCMD is a general macro to execute TOPSPIN commands for which no dedicated macro exists. For most TOPSPIN commands a dedicated macro does exist and we strongly recommend to:

Use dedicated macros whenever available

Note that XCMD executes CPR_exec with the option WAIT_TERM. If you want to use the options CONT_EX or WAIT_START, you must use CPR_exec.

If you want to check whether or not XCMD was executed successfully, you can check the value of AUERR (NORM_TERM or ERR_TERM).

EXAMPLE

The following AU program gets the foreground dataset, opens the acquisition parameter editor (eda) and runs an acquisition and Fourier transform:

    XCMD("sendgui eda")
    ZG
    FT
    QUIT

SEE ALSO

CPR_exec - C-function for executing special TOPSPIN commands.
WAIT_UNTIL

NAME

WAIT_UNTIL - hold the AU program until the specified date and time

SYNTAX

int WAIT_UNTIL(int hour, int minute, int day, int month)

DESCRIPTION

The function WAIT_UNTIL waits in an AU program until the specified date has been reached. The variables are internally converted to seconds. Every sixty seconds, the function checks whether the current date matches with the selected date. This function basically allows to program an event or command to start at a certain date rather than waiting for a certain time until something is executed.

EXAMPLE

Wait in the AU program until the 31st of October, 6 pm, and then continue:

WAIT_UNTIL(18,0,31,10)

SEE ALSO

ssleep - pause in an AU program for a certain number of seconds
Chapter 4

TOPSPIN Interface functions

AU programs are normally used to execute a series of acquisition or processing commands. For these commands you can use dedicated AU macros like ZG and FT. Less common is the use of TOPSPIN Java interface commands in AU programs. You can, however, do that with the XCMD or CPR_Exec macros and the command sendgui. Two examples are

```
XCMD("sendgui eda")
```

to display the acquisition parameters

```
CPR_exec("sendgui .vr", WAIT_START);
```

to perform a vertical reset of the current dataset

Note that XCMD is the same as CPR_exec with WAIT_TERM.

This can be used for all TOPSPIN interface commands like Data window Tabs,
Menu entries and Toolbar buttons. Here are some examples

<table>
<thead>
<tr>
<th>TOPSPIN Interface</th>
<th>TOPSPIN Command</th>
<th>AU statement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Menus</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>File ’ reopen</em></td>
<td>reopen</td>
<td>XCMD(&quot;sendgui reopen&quot;)</td>
</tr>
<tr>
<td><em>File ’ Close</em></td>
<td>close</td>
<td>XCMD(&quot;sendgui close&quot;)</td>
</tr>
<tr>
<td><em>Window ’ New Window</em></td>
<td>newwin</td>
<td>XCMD(&quot;sendgui newwin&quot;)</td>
</tr>
<tr>
<td><strong>Data Window Tabs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Spectrum</em></td>
<td>spec</td>
<td>XCMD(&quot;sendgui spec&quot;)</td>
</tr>
<tr>
<td><em>ProcPars</em></td>
<td>edp</td>
<td>XCMD(&quot;sendgui edp&quot;)</td>
</tr>
<tr>
<td><em>Title</em></td>
<td>edti</td>
<td>XCMD(&quot;sendgui edti&quot;)</td>
</tr>
<tr>
<td><strong>Toolbar buttons</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.vr</td>
<td></td>
<td>XCMD(&quot;sendgui .vr&quot;)</td>
</tr>
<tr>
<td>.zi</td>
<td></td>
<td>XCMD(&quot;sendgui .zi&quot;)</td>
</tr>
<tr>
<td>.ov</td>
<td></td>
<td>XCMD(&quot;sendgui .ov&quot;)</td>
</tr>
</tbody>
</table>

Table 4.1
Chapter 5

Macros changing the current AU dataset

This chapter contains a description of all AU macros which can be used to change the current AU dataset, i.e. the dataset on which subsequent AU statements operate.
SETCURDATA

NAME

SETCURDATA - make the current AU dataset available for subsequent AU statements

SYNTAX

SETCURDATA

DESCRIPTION

SETCURDATA makes the current AU dataset, i.e. the dataset defined by the data path variables disk, user, type, name, expno and procno, available for subsequent AU commands. Normally, you do not need to enter SETCURDATA because it is automatically called by macros which operate on datasets before they perform their actual task. Furthermore, the macros DATASET and GETDATASET, which change the current AU dataset, automatically call SETCURDATA after they performed their actual task. In some cases, however, SETCURDATA must be specified explicitly in the AU program. For example, the macros IEXPNO and IPROCNO change the current AU dataset, but do not call SETCURDATA. If they are followed by a CPR_exec or any C-statement which access the current AU dataset, then you must precede that statement with SETCURDATA.

EXAMPLE

This example shows the part of the library AU program multizg which calculates the total experiment time of all acquisitions performed by this AU program:

```c
int expTime;
static void PrintExpTime();
....
expTime = 0;
TIMES(i1)
    SETCURDATA;
    expTime += CalcExpTime() + 4;
```
Macros changing the current AU dataset

```plaintext
IEXPNO;
END
DEXPNO;
....
QUIT

Note that IEXPNO is followed by SETCURDATA in the next cycle of the loop.

SEE ALSO

DATASET - set the current AU dataset
IEXPNO - increase the experiment number by one
```
DATASET

NAME

DATASET - set the current AU dataset

SYNTAX

DATASET(char *name, int expno, int procno, char *disk, char *user)

DESCRIPTION

The macro DATASET sets the current AU dataset. All data path variables name, expno, procno, disk and user must be specified as arguments. Subsequent AU commands will operate on this dataset.

EXAMPLE

The following AU program first gets the foreground dataset, then selects a new dataset and runs an acquisition:

```
char newname[20];
strcpy(newname, "glycerine");
DATASET(newname, expno, 3, disk, "peter")
ZG
QUIT
```

The data path variables in this example are entered in the following way:

- expno and disk keep the values of the current dataset
- name gets the value of newname, a variable defined in this AU program
- procno and user get the values 3 and peter, respectively, which are entered as constants

SEE ALSO

GETDATASET - prompt the user to specify a new dataset
DATASET2 - set the second dataset
IEXPNO - increase the experiment number by one
DATASET2, DATASET3

NAME
DATASET2 - set the second AU dataset
DATASET3 - set the third AU dataset

SYNTAX
DATASET2(char *name, int expno, int procno, char *disk, char *user)
DATASET3(char *name, int expno, int procno, char *disk, char *user)

DESCRIPTION
The macro DATASET2 sets the second AU dataset. The current (first) AU dataset is not affected by this macro. DATASET2 is typically used in combination with algebra macros, like ADD or MUL, which operate on the second and third dataset.

EXAMPLE
The following AU program gets the foreground dataset, adds the spectra of the next processing number and the one after that and stores the result into the current dataset:

    DATASET2(name, expno, procno+1, disk, user)
    DATASET3(name, expno, procno+2, disk, user)
    ADD
    QUIT

SEE ALSO
DATASET - set the current AU dataset
GETDATASET - prompt the user to specify a new dataset
GETDATASET

NAME
GETDATASET - prompt the user to specify a new dataset

SYNTAX
GETDATASET

DESCRIPTION
The macro GETDATASET prompts the user to specify a new dataset. A dialogue is opened and the user is requested to enter the data path variables name, expno, procno, user and disk. Subsequent AU commands will operate on this dataset. GETDATASET can be used anywhere in an AU program but, since it requires user input, should not be used in fully automated sequences.

NOTE
GETDATASET is not used very often. In AU programs, datasets are usually changed without user interaction, e.g. with the macros DATASET, IEXPNO etc.

EXAMPLE
The following AU program gets the foreground dataset, prompts the user to specify a new dataset and then processes this dataset:

   GETDATASET
   EFP
   QUIT

SEE ALSO
DATASET - set the current AU dataset
IEXPNO - increase the experiment number by one
IPROCN0 - increase the processing number by one
IEXPNO

NAME
IEXPNO - increase the experiment number by one

SYNTAX
IEXPNO

DESCRIPTION
The macro IEXPNO increases the experiment number of the current AU dataset by one. In fact, the value of the data path variable expno is incremented by one. Subsequent macros will operate on this new expno. IEXPNO is typically used in AU programs which run a series of acquisitions on datasets with the same name and successive expnos.

EXAMPLE
The following AU program gets the foreground dataset and runs acquisitions on eight successive expnos:

TIMES(8)
ZG
IEXPNO
END
QUIT

NOTE
IEXPNO must be followed by a SETCURDATA if the AU program continues with an explicit CPR_exec or C-statement (see SETCURDATA).

SEE ALSO
DEXPNO - decrease the experiment number by one
REXPNO - set the experiment number to the specified value
IPROCNO - increase the processing number by one
DATASET- set the current AU dataset
DEXPNO

NAME

DEXPNO - decrease the experiment number by one

SYNTAX

DEXPNO

DESCRIPTION

The macro DEXPNO decreases the experiment number of the current AU dataset by one. In fact, the value of the data path variable \textit{expno} is decremented by one. Subsequent macros will operate on this new \textit{expno}. DEXPNO is typically used after a loop which includes an IEXPNO at the end, to revert the effect of the last (unnecessary) IEXPNO.

EXAMPLE

The following AU program gets the foreground dataset, runs acquisitions on eight successive \textit{expnos} and displays the data of the last \textit{expno}:

\begin{verbatim}
    TIMES(8)
    ZG
    IEXPNO
    END
    DEXPNO
    VIEWDATA
    QUIT
\end{verbatim}

Note that DEXPNO must be followed by a SETCURDATA if the AU program continues with an explicit CPR\_exec or C-statement (see SETCURDATA).

SEE ALSO

IEXPNO - increase the experiment number by one
REXPNO - set the experiment number to the specified value
DPROCNO - decrease the processing number by one
REXPNO

NAME

REXPNO - set the experiment number to the specified value

SYNTAX

REXPNO(int number)

DESCRIPTION

The macro REXPNO sets the experiment number of the current AU dataset to the specified value. In fact, the value of the data path variable expno is set. Subsequent macros will operate on this new expno.

EXAMPLE

The following AU program gets the foreground dataset, runs acquisitions on eight successive expnos then sets the current AU dataset back to the first expno and Fourier transforms it:

   i1=expno;
   TIMES(8)
       ZG
       IEXPNO
   END
   REXPNO(i1)
   FT
   QUIT

Note that REXPNO must be followed by a SETCURDATA if the AU program continues with an explicit CPR_exec or C-statement (see SETCURDATA).

SEE ALSO

IEXPNO - increase the experiment number by one
DEXPNO - decrease the experiment number by one
RPROCNO - set the processing number to the specified value
IPROCNO

NAME

IPROCNO - increase the processing number by one

SYNTAX

IPROCNO

DESCRIPTION

The macro IPROCNO increases the processing number of the current AU dataset by one. In fact, the value of the data path variable procno is incremented by one. Subsequent macros will operate on this new procno. IPROCNO is typically used in an AU program which processes a series of datasets with same name and expno and successive procnos.

EXAMPLE

The following AU program runs Fourier transforms on eight successive procnos:

    TIMES(8)
    FT
    IPROCNO
    END
    QUIT

Note that IPROCNO must be followed by a SETCURDATA if the AU program continues with an explicit CPR_exec or C-statement (see setcurdata).

SEE ALSO

DPROCNO - decrease the processing number by one
RPROCNO - set the processing number to the specified value
IEXPNO - increase the experiment number by one
DPROCNO

NAME
DPROCNO - decrease the processing number by one

SYNTAX
DPROCNO

DESCRIPTION
The macro DPROCNO decreases the processing number of the current AU dataset by one. In fact, the value of the data path variable procno is decremented by one. Subsequent macros will operate on this new procno. DPROCNO is typically used after a loop which includes an IPROCNO at the end, to revert the effect of the last (unnecessary) IPROCNO.

EXAMPLE
The following AU program gets the foreground dataset, runs a Fourier transform on eight successive procnos and displays the data of the last procno:

TIMES(8)
    FT
    IPROCNO
END
DPROCNO
VIEWDATA
QUIT

Note that DPROCNO must be followed by a SETCURDATA if the AU program continues with an explicit CPR_exec or C-statement (see setcurdata).

SEE ALSO
IPROCNO - decrease the experiment number by one
RPROCNO - set the processing number to specified value
DEXPNO - decrease the experiment number by one
RPROCNO

NAME

RPROCNO - set the processing number to the specified value

SYNTAX

RPROCNO(int number)

DESCRIPTION

The macro RPROCNO changes the current AU dataset by setting the processing number to the specified value. In fact, the value of the data path variable procno is set. Subsequent macros will then operate on this new procno.

EXAMPLE

The following AU program gets the foreground dataset and runs a Fourier transform on eight successive procnos. Then the current AU dataset is set back to the first procno which is then phase corrected:

i1=procno;
TIMES(8)
   FT
   IPROCNO
END
RPROCNO(i1)
APK
QUIT

Note that RPROCNO must be followed by a SETCURDATA if the AU program continues with an explicit CPR_exec or C-statement (see SETCURDATA).

SEE ALSO

IPROCNO - increase the processing number by one
DPROCNO - decrease the processing number by one
REXPNO - set the experiment number to the specified value
VIEWDATA

NAME

VIEWDATA - Show the current AU program dataset in new window

SYNTAX

VIEWDATA

DESCRIPTION

The macro VIEWDATA shows the current AU program dataset in a new
window or activates the window that contains this dataset. VIEWDATA is
used whenever the current AU dataset is changed within the AU pro-
gram, i.e. with DATASET, IEXPNO etc. and this dataset must be shown in
TOPSPIN.

EXAMPLE

The following AU program gets the foreground dataset, increases the
processing number and performs a Fourier transform storing the spec-
trum in this processing number. The spectrum is then shown in TOPSPIN:

    IPROCNO
    FT
    VIEWDATA
    QUIT

SEE ALSO

VIEWDATA_SAMEWIN - show the current dataset in the current window
GETDATASET - prompt the user to specify a new dataset
DATASET - set the current AU dataset
IEXPNO - increase the experiment number by one
IPROCNO - increase the processing number by one
VIEWDATASAMEWIN

NAME

VIEWDATASAMEWIN - Show the current AU program dataset in the current window

SYNTAX

VIEWDATASAMEWIN

DESCRIPTION

The macro VIEWDATASAMEWIN shows the current AU program dataset in the current window that contains this dataset. It is used as an alternative to VIEWDATA.

EXAMPLE

The following AU program gets the foreground dataset, increases the processing number and performs a Fourier transform storing the spectrum in this processing number. The spectrum is then shown in TOPSPIN:

\[
\begin{align*}
\text{IPROCNO} \\
\text{FT} \\
\text{VIEWDATASAMEWIN} \\
\text{QUIT}
\end{align*}
\]

SEE ALSO

VIEWDATA - show the current dataset in new window
GETDATASET - prompt the user to specify a new dataset
DATASET - set the current AU dataset
IEXPNO - increase the experiment number by one
IPROCNO - increase the processing number by one
Chapter 6

Macros copying datasets

This chapter contains a description of all AU macros which can be used to copy the current AU dataset or parts of it to a new dataset.
WRA

NAME
WRA - copy the raw data to the specified experiment number

SYNTAX
WRA(int expno)

DESCRIPTION
The macro WRA copies the raw data, including the acquisition and processing parameters of the current AU dataset to a new experiment number. It does not copy the processed data.

EXAMPLE
The following AU program gets the foreground dataset and copies the raw data to eight successive experiment numbers, starting with expno 11:

```au
i1=11;
TIMES(8)
   WRA(i1)
   i1++;
END
QUIT
```

SEE ALSO
WRP - copy the processed data to the specified processing number
WRPA - copy the raw and processed data to the specified dataset
WRP

NAME

WRP - copy the processed data to the specified processing number

SYNTAX

WRP(int procno)

DESCRIPTION

The macro WRP copies the processed data, including the processing parameters of the current AU dataset, to the specified processing number.

EXAMPLE

The following AU program gets the foreground dataset and copies the processed data to eight successive processing numbers, starting with procno 11:

\[
\begin{align*}
&i1=11; \\
&TIMES(8) \\
&\quad WRP(i1) \\
&\quad i1++; \\
&END \\
&QUIT
\end{align*}
\]

SEE ALSO

WRA - copy the raw data to the specified experiment number
WRPA - copy the raw and processed data to the specified dataset
WRPA

NAME
WRPA - copy the raw and processed data to the specified dataset

SYNTAX
WRPA(char *name, int expno, int procno, char *disk, char *user)

DESCRIPTION
The macro WRPA copies the raw and processed data of the current AU dataset to the specified dataset. WRPA takes 5 arguments, name, expno, procno, disk and user, i.e. the data path variables which define the dataset path. You can set one, several, or all of these variables to new values in order to define the destination dataset. You can, for instance, archive your data to an external medium by changing the value of the variable disk and leaving the other path variables the same.

EXAMPLE
The following AU program copies the current dataset to an external disk drive E:/:

    WRPA(name, expno, procno, "E:/", user)
    QUIT

SEE ALSO
WRA - copy the raw data to the specified experiment number
WRP - copy the processed data to the specified processing number
Chapter 7
Macros handling rows/columns

This chapter contains a description of all AU macros which can be used to read (write) rows or columns from (to) a 2D dataset and AU macros that can be used to read rows or planes from 3D raw data.
RSR

NAME

RSR - read a row from a 2D spectrum and store it as a 1D spectrum

SYNTAX

RSR(int row, int procno)

DESCRIPTION

The macro RSR reads a row from a 2D spectrum and stores it as a 1D spectrum. It can be used in the following ways:

- specified with procno > 0, executed on a 2D dataset
  the specified row is stored under the current dataname, the current expno and the specified procno.

- specified with procno = -1, executed on a 2D dataset
  the specified row is stored under dataset ~TEMP/1/pdata/1

- specified with procno > 0, executed on a 1D dataset
  the specified row is read from a 2D dataset that resides under the current dataname, the current expno and the specified procno.

- specified with procno = -1, executed on a 1D dataset
  the specified row is read from the 2D dataset from which the current 1D dataset was extracted (as defined in the file `used_from`).

EXAMPLE

The following AU program gets a 2D dataset and processes it. Then it reads row 16 and stores that under procno 999:

```au
DATASET("my_2D_data", 1, 1, "C:/bio", "guest")
XFB
RSR(16, 999)
QUIT
```

SEE ALSO

RSC - read a column from a 2D spectrum and store it as a 1D spectrum
RSC

NAME

RSC - read a column from a 2D spectrum and store it as a 1D spectrum

SYNTAX

RSC(int column, int procno)

DESCRIPTION

The macro RSC reads a column from a 2D spectrum and stores it as a 1D spectrum. It can be used in the following ways:

- specified with procno > 0, executed on a 2D dataset
  the specified column is stored under the current dataname, the current expno and the specified procno.

- specified with procno = -1, executed on a 2D dataset
  the specified column is stored under dataset ~TEMP/1/pdata/1

- specified with procno > 0, executed on a 1D dataset
  the specified column is read from a 2D dataset that resides under the current dataname, the current expno and the specified procno.

- specified with procno = -1, executed on a 1D dataset
  the specified column is read from the 2D dataset from which the current 1D dataset was extracted (as defined in the file used_from).

EXAMPLE

The following AU program gets a 2D dataset and processes it in the F2 dimension. Then it reads column 128 and processes the resulting 1D dataset:

```
DATASET("my_2D_data", 1, 1, "C:/bio", "guest")
XF2
RSC(128, 10)
RPROCNO(10)
EF
QUIT
```
SEE ALSO

RSR - read a row from a 2D spectrum and store it as a 1D spectrum
WSC - replace a column of a 2D spectrum by a 1D spectrum
WSR

NAME
WSR - replace a row of a 2D spectrum by a 1D spectrum

SYNTAX
WSR(int row, int procno, int expno, char *name, char *user, char *disk)

DESCRIPTION
The macro WSR replaces a row of a 2D spectrum by a 1D spectrum. It can be used in the following ways:

• executed on a 1D dataset
  the specified row of the specified dataset (must 2D data) is replaced by the current 1D data.

• executed on a 2D dataset
  the specified row of the current 2D dataset is replaced by the specified dataset (must be 1D data)

EXAMPLE
The following AU program gets a 2D dataset, reads row 16, phase corrects this row and writes it back to the 2D data:

  DATASET("my_2D_data", 1, 1, "C:/bio", "guest")
  XFB
  RSR(16, 999)
  RPROCNO(999)
  APK
  WSR(16, 1, expno, name, user, disk)
  QUIT

SEE ALSO
WSC - replace a column of a 2D spectrum by a 1D spectrum
RSR - read a row from a 2D spectrum and store it as a 1D spectrum
WSC

NAME
WSC - replace a column of a 2D spectrum by a 1D spectrum

SYNTAX
WSC(int column, int procno, int expno, char *name, char *user, char *disk)

DESCRIPTION
The macro WSC replaces a column of a 2D spectrum by a 1D spectrum. It can be used in the following ways:

• executed on a 1D dataset
  the specified column of the specified dataset (must 2D data) is replaced by the current 1D data.

• executed on a 2D dataset
  the specified column of the current 2D dataset is replaced by the specified dataset (must be 1D data)

EXAMPLE
The following AU program gets a 2D dataset, reads column 16, phase corrects this column and writes it back to the 2D data:

  DATASET("my_2D_data", 1, 1, "C:/bio", "guest")
  RSC(16, 999)
  RPROCNO(999)
  APK
  WSC(16, 1, expno, name, user, disk)
  QUIT

SEE ALSO
WSR - replace a row of a 2D spectrum by a 1D spectrum
RSC - read a column from a 2D spectrum and store it as a 1D spectrum
RSER

NAME

RSER - read a row from 2D or 3D raw data and store it as a 1D FID

SYNTAX

RSER(int row, int expno, int procno)

DESCRIPTION

The macro RSER reads a row from 2D or 3D raw data and stores it as a 1D fid. It can be used in the following ways:

- specified with expno > 0, executed on a 2D dataset
  the specified row is stored under the current dataname and the specified expno. Processing parameters are stored under procno 1.

- specified with expno = -1, executed on a 2D dataset
  the specified row is stored under dataset ~TEMP/1/pdata/1

- specified with expno > 0, executed on a 1D dataset
  the specified row is read from a 2D raw data that resides under the current dataname and the specified expno. Processing parameters are read from procno 1.

- specified with expno = -1, executed on a 1D dataset
  the specified row is read from the 2D dataset from which the current 1D dataset was extracted (as defined in the file used_from).

EXAMPLE

The following AU program splits 2D raw data into single fids that are stored in successive expnos:

```au
int td;
FETCHPAR1S("TD",&td)
i1=0;
TIMES(td)
  i1 ++;
  RSER(i1,i1+expno,1)
```
END
QUITMSG("--- splitser finished ---")

Note that this is the AU program \texttt{splitser} that is delivered with \texttt{TOPSPIN}.

\textbf{SEE ALSO}

- \texttt{WSER} - replace a row of 2D raw data by 1D raw data
- \texttt{RSER2D} - read a plane from 3D raw data and store it as 2D raw data
- \texttt{RSR} - read a row from a 2D spectrum and store it as a 1D spectrum
WSER

NAME

WSER - replace a row of 2D raw data by 1D raw data

SYNTAX

WSER(int row, char *name, int expno, int procno, char *disk, char *user)

DESCRIPTION

The macro WSER replaces a row of 2D raw data by 1D raw data. It can be used in the following ways:

- executed on a 1D dataset
  the specified row of the specified dataset (must be 2D data) is replaced by the current 1D data.
- executed on a 2D dataset
  the specified row of the current 2D dataset is replaced by the specified dataset (must be 1D data)

EXAMPLE

The following AU program writes a number of 1D fids that are stored under the same data name and incremental expnos to 2D raw data:

```c
int ne, exp1, proc1;
char nm1[20];
ne=1; exp1=1; proc1=1;
strcpy(nm1, name);
GETSTRING("Enter name of 1D series: ", nm1)
GETINT("Enter starting EXPNO: ", exp1)
GETINT("Enter starting PROCNO: ", proc1)
GETINT("Enter # of Fids: ", ne)
USECURPARS
TIMES(ne)
  WSER(loopcount1+1, nm1, exp1, proc1, disk, user)
   exp1++;
END
```
QUIT

Note that this is the AU program *fidtoser* that is delivered with TOPSPIN.

**SEE ALSO**

RSER - read a row from 2D or 3D raw data and store it as a 1D FID
WSR - replace a row of a 2D spectrum by a 1D spectrum
WSC - replace a column of a 2D spectrum by a 1D spectrum
RSER2D

NAME

RSER2D - read a plane from 3D raw data and store it as 2D pseudo raw data

SYNTAX

RSER2D(char *direction, int plane, int expno, int procno)

DESCRIPTION

The macro RSER2D reads a plane from 3D raw data and stores it as 2D pseudo raw data. The first argument, the plane direction can be "s23" or "s13" for the F2-F3 or F1-F3 direction, respectively. The specified plane is stored under the current data name, the specified expno and the specified procno.

EXAMPLE

The following AU program gets a 3D dataset, reads the F2-F3-plane 64 and stores that under expno 11. It then switches to the output 2D dataset and processes it.

    DATASET("my_3D_data", 1, 1, "C:/bio", "guest")
    RSER2D("s23", 64, 11)
    REXPNO(11)
    XFB
    QUIT

SEE ALSO

RSER - read a row from 2D or 3D raw data and store it as a 1D FID
WSER - replace a row of 2D raw data by 1D raw data
This chapter contains a description of all AU macros which can be used to convert TOPSPIN data. This includes the conversion of Bruker Aspect 2000/3000 data, Varian data and Jeol data to TOPSPIN data format as well as the conversion of TOPSPIN data to JCAMP-DX.
TOJDX, TOJDX5

NAME
TOJDX - convert a dataset to JCAMP-DX 6.0 format
TOJDX5 - convert a dataset to JCAMP-DX 5.0 format

SYNTAX
TOJDX(char *path, int type, int mode, char *title, char *origin, char *owner)

TOJDX5(char *path, int type, int mode, char *title, char *origin, char
*owner)

DESCRIPTION
The macro TOJDX converts the current AU data to standard JCAMP-DX 6.0 format. It takes 6 arguments:

1. the pathname of the output file, e.g. /tmp/data1.dx
2. the output type: enter a number between 0 and 6, where:
   0 = FID (default)
   1 = real spectrum
   2 = complex spectrum
   3 = parameter files
   4 = raw data + real and imaginary processed data
   5 = raw data + real and imaginary processed data of all PROCNO’s under the current EXPNO
   6 = raw data + real and imaginary processed data of all EXPNO’s under the current NAME
3. the compression mode: enter 0, 1, 2 or 3
   0=FIX, 1=PACKED, 2=SQUEEZED, 3=DIF/DUP (default)
4. the title as it appears in the output file: enter a character-string
5. the origin as it appears in the output file: enter a character-string
6. the owner as it appears in the output file: enter a character-string

If "*" is entered as an argument, then the default value is used.
Note that the macro TOJDX5 only supports the output types 0, 1, 2 and
3.

EXAMPLE

The following AU program gets the foreground dataset and performs a conversion to JCAMP on 5 successive experiment numbers. The name of the JCAMP file contains the name and expno of the corresponding TOPSPIN dataset.

\[
\text{TIMES(5)}
\text{printf(text,"C:/TEMP/%s_%d.dx", name, expno);} \\
\text{TOJDX(text, 0, 3, ",", ",", ",")} \\
\text{IEXPNO} \\
\text{END} \\
\text{QUIT}
\]

SEE ALSO

FROMJDX - convert a JCAMP-DX file to TOPSPIN data format
FROMJDX

NAME
FROMJDX - convert a JCAMP-DX file to TOPSPIN data format

SYNTAX
FROMJDX(char *input-file)

DESCRIPTION
The macro FROMJDX converts a JCAMP-DX file to TOPSPIN data format. It takes one argument; the pathname of the input file, e.g.
/tmp/data1.dx

FROMJDX can convert 1D and 2D data.

EXAMPLE
The following AU program converts all files with the extension .dx in the directory C:/TEMP to a TOPSPIN dataset:

char **listfile;
i1 = getdir("C:/TEMP",&listfile,"*.dx");
TIMES(i1)
   sprintf(text, "C:/TEMP/%s", listfile[loopcount1]);
   FROMJDX(text)
END
QUIT

SEE ALSO
TOJDX - convert a dataset to JCAMP-DX format
getdir - get all file names and/or directory names within a directory
VCONV

NAME

VCONV - convert a Varian dataset to Bruker TOPSPIN format

SYNTAX

VCONV(char *v-name, char *x-name, int expno, char *disk, char *user)

DESCRIPTION

The macro VCONV converts a Varian dataset to TOPSPIN data format. It takes 5 parameters:

1. the name of the input Varian dataset
2. the name of the output TOPSPIN dataset
3. the experiment number of the output TOPSPIN dataset
4. the disk unit of the output TOPSPIN dataset
5. the user of the output TOPSPIN dataset

EXAMPLE

The following AU program converts a Varian dataset to TOPSPIN format:

VCONV("pinen_h.fid", "pinen_h", 1, "C:/bio", "joe")
QUIT

Note that VCONV searches for the input data file in the directory defined by the environment variable VNMR. Assume the file resides in C:/bio. You can set VNMR from the TOPSPIN command line with:

env set VNMR=c:/bio

or inside the AU program with:

CPR_exec("env set VNMR=C:/bio", WAIT_TERM);

SEE ALSO

JCONV - convert a Jeol dataset to Bruker TOPSPIN format
JCONV

NAME
JCONV - convert a Jeol dataset to Bruker TOPSPIN format

SYNTAX
JCONV(char *j-name, char *x-name, int expno, char *disk, char *user)

DESCRIPTION
The macro JCONV converts a Jeol dataset to TOPSPIN data format. It takes 5 parameters:

1. the name of the input Jeol dataset
2. the name of the output TOPSPIN dataset
3. the experiment number of the output TOPSPIN dataset
4. the disk unit of the output TOPSPIN dataset
5. the user of the output TOPSPIN dataset

Note that JCONV searches for the input data file in the directory defined by the environment variable JNMR. Assume the file resides in C:/bio. You can set JNMR from the TOPSPIN command line with:

```env set JNMR=c:/bio```

or inside the AU program with:

```CPR_exec("env set JNMR=C:/bio", WAIT_TERM);```

EXAMPLE
The following AU program converts a Jeol dataset to TOPSPIN format:

```JCONV("gx400h.gxd", "gx400h", 1, "C:/bio", "joe")```

QUIT

SEE ALSO
VCONV - convert a Varian dataset to Bruker TOPSPIN format
This chapter contains a description of AU macros which can be used to get and store TOPSPIN parameters. Parameters are subdivided in acquisition, processing, output and plot parameters. Furthermore, they exist in two different forms; as foreground and status parameters. Finally, multi-dimensional datasets have parameter sets for each dimension. Different AU macros are available for getting and storing parameters of all categories, forms or dimensions.
FETCHPAR

NAME

FETCHPAR - get an acquisition, processing or output parameter

SYNTAX

FETCHPAR(par, &val)

DESCRIPTION

The macro FETCHPAR gets the value of a foreground parameter and stores it into an AU variable. This AU variable can then be used in subsequent AU statements. FETCHPAR allows to get acquisition parameters (eda) and processing parameters (edp). It is typically used to check or modify a parameter prior to an acquisition or processing statement.

The macro FETCHPAR takes two arguments:

1. the name of the parameter
2. the AU variable into which the parameter value will be stored

There are two important things to be considered:

1. The type of the AU variable must be the same as the type of the parameter (see Chapter 14).
2. The second argument must be specified as the variable's address, i.e. it must be prepended with the '&' character. This, however, does not count for a text variable since a text variable is already an address.

FETCHPAR works on 1D, 2D or 3D datasets and always gets a parameter of the first dimension (F2 for 1D, F2 for 2D and F3 for 3D).

The handling of the macros FETCHPAR1, FETCHPAR3, FETCHPARM, FETCHT1PAR and FETCHDOSYPAR is equivalent to the handling of FETCHPAR.

EXAMPLES

The following AU program gets the value of the processing parameter SI and processes the data 4 times, each time doubling the spectrum size
and storing the data in successive processing numbers:

    FETCHPAR("SI", &i1)
    TIMES(4)
    EFP
    IPROCNO
    i1 = i1*2;
    STOREPAR("SI", i1)
    END
    QUIT

The following AU statements get the values of the acquisition parameter DW and the processing parameter STSI and stores them in the prede fined variables f1 and i1, respectively. Then it gets value of the parameter ABSF1 and stores it in the user defined variable leftlimit.

    float leftlimit;
    ...
    FETCHPAR("DW", &f1)
    FETCHPAR("STSI", &i1)
    FETCHPAR("ABSF1", &leftlimit)

SEE ALSO

    FETCHPARS - get a status parameter
    FETCHPARN - get a parameter from specified direction
    STOREPAR - store an acquisition, processing or output parameter
FETCHPARS

NAME

FETCHPARS - get a status parameter (acquisition and processing)

SYNTAX

FETCHPARS(par, &val)

DESCRIPTION

The macro FETCHPARS gets the value of a status parameter and stores it into an AU variable. This AU variable can then be used in subsequent AU statements. Acquisition status parameters are set by acquisition commands and describe the status of the dataset after acquisition. Note that the status parameters (dpa) describe what really happened and that this is sometimes different from what was set up before the acquisition as acquisition parameters (eda). For example, the status NS is smaller than originally specified when an acquisition was halted prematurely. Any AU program statement which follows an acquisition command and evaluates acquisition parameters must read status parameters. Therefore, FETCHPARS is typically used after acquisition or processing statements, for example for error or abort conditions (see example below).

The macro FETCHPARS takes two arguments:

1. the name of the parameter
2. the AU variable into which the value is value will be stored

There are two important things to be considered:

1. The type of the AU variable must be the same as the type of the parameter (see Chapter 14).
2. The second argument must be specified as the variable's address, i.e. it must be prepended with the '&' character. This, however, does not count for a text variable since a text variable is already an address.

The handling of the macros FETCHPARS1 and FETCHPARS3 is equivalent to the handling of FETCHPARS.
EXAMPLE

The following AU program performs a series of acquisitions on the same dataset until a minimum signal/noise is reached. In a loop 8 scans are acquired, Fourier transformed and phase corrected. Then the signal/noise of the spectrum is calculated and compared with the minimum value. If the minimum signal/noise was not reached yet, 8 more scans are accumulated etc. A maximum of 8000 scans is acquired. After the acquisition has been stopped, the total number of actually acquired scans is displayed.

STOREPAR("NS", 8)
GETFLOAT("Please enter the minimum signal/noise", f1)
ZG
TIMES(1000)
   FT
   APK
   SINO
   FETCHPARS("SINO", f2)
   if (f1 >= f2)
      break;
   GO
END
FETCHPARS("NS", i1)
Proc_err (DEF_ERR_OPT,"Acquisition stopped after %d scans", i1);
QUIT

SEE ALSO

FETCHPAR - get an acquisition, processing or output parameter
FETCHPARN - get a status parameter from specified direction
STOREPARS - store a status parameter (acquisition and processing)
STOREPAR

NAME
STOREPAR - store an acquisition, processing or output parameter

SYNTAX
STOREPAR(par, val)

DESCRIPTION
The macro STOREPAR stores the value of an AU variable into a parameter. This AU variable can then be used in subsequent AU statements. STOREPAR can be used for acquisition parameters (eda) and processing parameters (edp). It is typically used to set parameters prior to an acquisition or processing statement. STOREPAR takes two arguments:

1. the name of the parameter
2. the value to be stored which can specified in two different forms:
   • as a constant
   • as the name of an AU variable

Important: the type of the parameter must be the same as the type of the constant or variable. (see Chapter 14).

NOTES
STOREPAR works on 1D, 2D or 3D datasets and always stores a parameter of the first dimension (F2 for 1D, F2 for 2D and F3 for 3D).

The handling of the macros STOREPAR1, STOREPAR3, STORET1PAR and STOREDOSYPAR is equivalent to the handling of STOREPAR.

EXAMPLE
The following AU program reads a standard parameter set, sets the pulse program and power level and asks the user for the number of scans. Then a dataset is acquired and processed according to these parameters.

RPAR("PROTON", "all")
STOREPAR("PULPROG", "zg30")
STOREPAR("PL 1", 10.0)
GETINT("Please enter the number of scans:", i1)
STOREPAR("NS", i1)
ZG
EFP
QUIT

SEE ALSO
STOREPARS - store a status parameter
STOREPARN - store a parameter to specified direction
FETCHPAR - get an acquisition, processing or output parameter
STOREPARN

NAME
STOREPAR - store a parameter to the specified direction

SYNTAX
STOREPARN(dir, par, val)

DESCRIPTION
TOPSPIN 2.1 and newer offers the macro STOREPARN. It works like STOREPAR except that it can be used for any direction of an n-dimensional dataset. STOREPARN takes three arguments:

1. the direction of the dataset
2. the name of the parameter
3. the value to be stored which can specified in two different forms:
   • as a constant
   • as the name of an AU variable

STOREPARN works on nD datasets of any dimension.

TOPSPIN 2.0 and older only supported AU parameter storage up to 3D, using the macros STOREPAR, STOREPAR1 and STOREPAR3. In TOPSPIN 2.1 and newer, these macros can still be used or they can be replaced by STOREPARN. Note that the direction specification for STOREPARN is different from STOREPAR/1/3.

For a 2D dataset:

F2 direction (acquisition direction):
STOREPAR(par, val)  or STOREPARN(2, par, val)

F1 direction:
STOREPAR1(par, val)  or STOREPARN(1, par, val)

For a 3D dataset:

F3 direction (acquisition direction):

STOREPAR(par, val) or STOREPARN(3, par, val)

F2 direction:
STOREPAR1(par, val) or STOREPARN(2, par, val)

F1 direction:
STOREPAR3(par, val) or STOREPARN(1, par, val)

SEE ALSO
STOREPAR - store a parameter in acquisition direction
STOREPARN - store a status parameter to specified direction
FETCHPARN - fetch a parameter from specified direction
STOREPARS

NAME

STOREPARS - store a status parameter (acquisition and processing)

SYNTAX

STOREPARS(par, val)

DESCRIPTION

The macro STOREPARS stores the value of an AU variable into a status parameter. This AU variable can then be used in subsequent AU statements. Status parameters are set by an acquisition or processing command and describe the status of the dataset after this acquisition or processing command. If the data are now manipulated by AU statements which do not update the status parameters, these must be set explicitly with STOREPARS. For example, if you add two fid’s with addfid, the total number of scans of the resulting dataset is not automatically updated. This must be done explicitly with STOREPARS.

The handling of the macros STOREPAR1S and STOREPAR3S is equivalent to the handling of STOREPARS.

EXAMPLE

The following AU program reads the foreground dataset, adds the fid of the next experiment number and the experiment number after that and stores the result in the foreground dataset. The number of scans of the original fid’s are added and stored in the status parameter NS of the resulting dataset.

```au
int expno_save;
DATASET2(name, expno+1, procno, disk, user)
DATASET3(name, expno+2, procno, disk, user)
expno_save=expno;
IEXPNO
FETCHPARS("NS", &i1)
IEXPNO
```
FETCHPARS("NS", &i2)
REXPNO(expno_save)
ADDFID
STOREPARS("NS", i1+i2)
QUIT

SEE ALSO

FETCHPARS - get a status parameter (acquisition and processing)
STOREPARN - store a status parameter to specified direction
STOREPAR - store an acquisition, processing or output parameter
RPAR

NAME
RPAR - read a parameter set to the current AU dataset

SYNTAX
RPAR(char *parset, char *typ)

DESCRIPTION
The macro RPAR reads a parameter set to the current AU dataset. This can be a standard Bruker parameter set or a user defined parameter set which was stored with WPAR. RPAR takes two arguments:

1. the name of the parameter set
2. the type of parameters which are to be read

The second argument can be:

- *acqu* for acquisition parameters (*eda*)
- *proc* for processing parameters (*edp*)
- *outd* for output parameters (*edp*)
- *all* for acquisition, processing, plot and output parameters

EXAMPLE
The following AU program reads the standard Bruker parameter set PROTON, sets the number of scans to 1024 and runs an acquisition:

RPAR("PROTON", "all")
STOREPAR("NS", 1024)
ZG
QUIT

SEE ALSO
WPAR - write the current dataset parameters to a parameter set
WPAR

NAME
WPAR - write the current dataset parameters to a parameter set

SYNTAX
WPAR(char *parset, char *typ)

DESCRIPTION
The macro WPAR writes the parameters of the current AU dataset to a parameter set. You can only write to user defined parameter sets. Bruker standard parameters sets cannot be overwritten. WPAR is typically used in AU programs to store a temporary parameter set. It takes two arguments:

1. the name of the parameter set
2. the type of parameters which are to be stored

The second argument can be:
- \textit{acqu} for acquisition parameters (\textit{eda})
- \textit{proc} for processing parameters (\textit{edp})
- \textit{outd} for output parameters
- \textit{all} for acquisition, processing, plot and output parameters

EXAMPLE
The following AU program reads the acquisition parameters of the Bruker standard parameter set PROTON, sets the number of scans, the frequency offset and time domain data size and writes the acquisition parameters to a temporary parameter set. It then performs 8 successive acquisitions with these parameters.

RPAR("PROTON", "all")
STOREPAR("NS", 16)
STOREPAR("O1", 766.23)
STOREPAR("TD",8192)
WPAR("partemp", "acqu")
TIMES(8)
ZG
IEXPNO
RPAR("partemp", "acqu")
END
QUIT

SEE ALSO

RPAR - read a parameter set to the current AU dataset
Chapter 10

Macros for Plot Editor/autoplot

This chapter contains a description of AU macros which can be used to plot data using Plot Editor portfolios and layouts. These include macros for the creation and definition of portfolios and for plotting to the printer, to a postscript file or enhanced metafile.

TOPSPIN 2.0 and newer also offer macros for automatic creation of Plot Editor layouts. Examples are LAYOUT_OBJ_1D and LAYOUT_ADD. These are described in a separate manual (see Help ’ Manuals ’ [Programming Manuals] Plot Layout Programming).
**AUTOPLLOT**

**NAME**

AUTOPLLOT - plot the current dataset according to a Plot Editor layout

**SYNTAX**

AUTOPLLOT

**DESCRIPTION**

The macro AUTOPLLOT plots the current dataset according to the Plot Editor layout that is defined by the parameter LAYOUT.

The Plot Editor layout can be:

- a standard layout that was delivered with TOPSPIN
- a user defined layout that was setup and stored from Plot Editor

Processing AU programs that contain the AUTOPLLOT macro can be used with one of the options a, e, h or t. They cause AUTOPLLOT to store the plot as a postscript file. For example, the AU program *proc_1d* can be enter as:

- **proc_1d** - prints to the printer defined in the layout
- **proc_1d a** - prints to a PDF file in the dataset procno
- **proc_1d e** - also prints to a postscript file in the dataset procno
- **proc_1d h** - also prints to a postscript file in the users home directory
- **proc_1d t** - also prints to a postscript file in the TEMP directory

Furthermore, you can use multiple arguments, e.g.:

- **proc_1d a e** - prints to a PDF file and to a postscript file in the dataset procno

(see also the header of the AU program *plot_to_file*)

**EXAMPLE**

This AU program processes the current 1D dataset and plots it according to the Plot Editor layout specified in *edp*:

EF
APK
SREF
ABS
AUTOPLOT
QUIT

SEE ALSO
AUTOPLOT_TO_FILE, AUTOPLOT_WITH_PORTFOLIO,
AUTOPLOT_WITH_PORTFOLIO_TO_FILE
AUTOPlot_TO_FILE

NAME
AUTOPlot_TO_FILE - as AUTOPlot but store the output into a file

SYNTAX
AUTOPlot_TO_FILE(filename)

DESCRIPTION
The macro AUTOPlot_TO_FILE plots the current dataset according to
the Plot Editor layout defined by the parameter LAYOUT. The output is
not sent to the printer but stored in the file that is specified as an argu-
ment. The argument is normally a full pathname. If it is not, the file is
stored in the TOPSPIN home directory.

If the filename has the extension .ps, the output is stored as a postscript
file. If (under Windows) it has the extension .emf, as in the example be-
low, the output will be stored as an enhanced metafile.

AUTOPlot_TO_FILE is actually a composite macro that consists of
several PORTFOLIO*/AUTOPlot* macros. This, however, is transpar-
ent to the user.

EXAMPLE
This AU program processes the current 1D dataset and plots it according
to the Plot Editor layout specified in edp. The result is stored in an en-
hanced metafile.

EF
APK
SREF
ABS
AUTOPlot_TO_FILE("C:/mydata.emf")
QUIT

SEE ALSO
AUTOPlot, AUTOPlot_WITH_PORTFOLIO
CREATE_PORTFOLIO

NAME
CREATE_PORTFOLIO - create a Plot Editor portfolio

SYNTAX
CREATE_PORTFOLIO(name)

DESCRIPTION
The macro CREATE_PORTFOLIO creates the Plot Editor portfolio that is specified as an argument. It takes one argument; the filename of the portfolio.

The argument is normally specified as a full pathname. If it is not, the portfolio is stored under the TOPSPIN home directory. If the specified file already exists, it is overwritten. Note that CREATE_PORTFOLIO creates the portfolio but does not insert any dataset specifications.

EXAMPLE
This AU program plots the current dataset according to the Plot Editor layout specified in \texttt{edp}. It is just a simple demonstration of the use of PORTFOLIO macros.

\begin{verbatim}
CREATE_PORTFOLIO("C:/temp/myPortfolio.por")
ADD_CURDAT_TO_PORTFOLIO
CLOSE_PORTFOLIO
AUTOPLLOT_WITH_PORTFOLIO
QUIT
\end{verbatim}

Note that this AU program does the same as the command \texttt{autoplot}.

SEE ALSO
ADD_TO_PORTFOLIO, CLOSE_PORTFOLIO
ADD_CURDAT_TO_PORTFOLIO

NAME
ADD_CURDAT_TO_PORTFOLIO - add the current dataset to the portfolio

SYNTAX
ADD_CURDAT_TO_PORTFOLIO

DESCRIPTION
The macro ADD_CURDAT_TO_PORTFOLIO adds the current dataset to the Plot Editor portfolio that was previously created with CREATE_PORTFOLIO.

EXAMPLE
This AU program plots two datasets, the current and next processing number of the current data name, according to the Plot Editor layout

    CREATE_PORTFOLIO("C:/temp/myPortfolio.por")
    ADD_CURDAT_TO_PORTFOLIO
    IPROCNO
    ADD_CURDAT_TO_PORTFOLIO
    CLOSE_PORTFOLIO
    AUTOPLOT_WITH_PORTFOLIO
    QUIT

SEE ALSO
CREATE_PORTFOLIO, CLOSEPORTFOLIO
ADD_TO_PORTFOLIO

NAME
ADD_TO_PORTFOLIO - add the specified dataset to the portfolio

SYNTAX
ADD_TO_PORTFOLIO(disk, user, name, expno, procno)

DESCRIPTION
The macro ADD_TO_PORTFOLIO adds a dataset to the portfolio that was previously created with CREATE_PORTFOLIO. The dataset to be added is completely specified by the five arguments of ADD_TO_PORTFOLIO. Note that these arguments can be constants (values) or variables.

EXAMPLE
This AU program plot two datasets according to the TOPSPIN layout. Note that the first dataset to be plotted is the so called second dataset (edc2), specified by the predefined dedicated variables disk2, user2 etc.

    CREATE_PORTFOLIO("/temp/myPortfolio.por")
    GETCURDATA2
    ADD_TO_PORTFOLIO(disk2, user2, name2, expno2, procno2)
    ADD_TO_PORTFOLIO("C:/ts", "guest", "mydata", 1, 3)
    CLOSE_PORTFOLIO
    AUTOPLOT_WITH_PORTFOLIO
    QUIT

SEE ALSO
ADD_CURDAT_TO_PORTFOLIO
CLOSE_PORTFOLIO

NAME
CLOSE_PORTFOLIO - closes the portfolio definition

SYNTAX
CLOSE_PORTFOLIO

DESCRIPTION
The macro CLOSE_PORTFOLIO closes the definition of the portfolio that was previously created with CREATE_PORTFOLIO. It must be used after the last ADD_CURDAT_TO_PORTFOLIO or ADD_TO_PORTFOLIO macro and before the first AUTOPLOT* macro.

EXAMPLE
This AU program plots the current dataset according to the TOPSPIN layout. It is just a simple demonstration of the use of PORTFOLIO macros.

    CREATE_PORTFOLIO("C:/temp/myPortfolio.por")
    ADD_CURDAT_TO_PORTFOLIO
    CLOSE_PORTFOLIO
    AUTOPLOT_WITH_PORTFOLIO
    QUIT

    Note that this AU program does the same as the command autoplot.

SEE ALSO
CREATE_PORTFOLIO, ADD_TO_PORTFOLIO
AUTOPLOT_WITH_PORTFOLIO

NAME
AUTOPLOT_WITH_PORTFOLIO - plot the dataset(s) of the current portfolio

SYNTAX
AUTOPLOT_WITH_PORTFOLIO

DESCRIPTION
The macro AUTOPLOT_WITH_PORTFOLIO plots the dataset(s) defined in the portfolio created with CREATE_PORTFOLIO according to the Plot Editor layout defined by the edp parameter LAYOUT.

EXAMPLE
This AU program plots the current dataset according to the TOPSPIN layout. It is just a simple demonstration of the use of PORTFOLIO macros.

CREATE_PORTFOLIO("C:/temp/myPortfolio.por")
ADD_CURDAT_TO_PORTFOLIO
CLOSE_PORTFOLIO
AUTOPLOT_WITH_PORTFOLIO
QUIT

Note that this AU program does the same as the command autoplot.

SEE ALSO
AUTOPLOT, AUTOPLOT_WITH_PORTFOLIO_TO_FILE
AUTOPLOT_WITH_PORTFOLIO_TO_FILE

NAME

AUTOPLOT_WITH_PORTFOLIO_TO_FILE - plot the dataset(s) of the current portfolio and store the output into a file

SYNTAX

AUTOPLOT_WITH_PORTFOLIO_TO_FILE(filename)

DESCRIPTION

The macro AUTOPLOT_WITH_PORTFOLIO_TO_FILE plots the dataset(s) defined in the Plot Editor portfolio that was previously created with CREATE_PORTFOLIO. The plot is made according to the layout defined by the parameter LAYOUT. The output is stored in the file that is specified as an argument to the macro. The argument is normally a full pathname. If it is not, the portfolio is stored under the TOPSPIN home directory.

If the filename has the extension .ps, as in the example below, the output will be stored as a postscript file. If (under Windows) it has the extension .emf, the output is stored as an enhanced metafile.

EXAMPLE

This AU program plots the current dataset according to the Plot Editor layout specified in edp and stores the result into a postscript file.

    CREATE_PORTFOLIO("C:/temp/myPortfolio.por")
    ADD_CURDAT_TO_PORTFOLIO
    CLOSE_PORTFOLIO

    AUTOPLOT_WITH_PORTFOLIO_TO_FILE("/users/guest/mydata.ps")

    QUIT

SEE ALSO AUTO

    PLOT_WITH_PORTFOLIO, AUTOPLOT_TO_FILE
Chapter 11

Macros prompting the user for input.

This chapter contains a description of all AU macros which can be used to prompt the user for input and store the input into an AU variable. Different macros are available for requesting integer, float, double or text values.
GETINT

NAME

GETINT - prompt the user to enter an integer value

SYNTAX

GETINT("Please enter an integer value : ", i1)

DESCRIPTION

The macro GETINT prompts the user to enter an integer value and stores this value into an integer variable. It can be used for various purposes, for example to set the value of a TOPSPIN (integer) parameter or to specify the number of cycles in an AU program loop. GETINT takes two arguments:

1. a text string which prompts the user to enter an integer value
2. an integer variable into which the value is stored

EXAMPLE

The following AU program prompts the user for the number of scans per acquisition and the number of experiments to be done:

GETINT("Please enter the number of scans: ", i1)
GETINT("Please enter the number of experiments: ", i2)
TIMES(i2)
    STOREPAR("NS", i1)
    ZG
    IEXPNO
END
QUIT

SEE ALSO

GETFLOAT - prompt the user to enter a float value
GETDOUBLE - prompt the user to enter a double value
GETSTRING - prompt the user to enter a text string
GETFLOAT, GETDOUBLE

NAME
GETFLOAT - prompt the user to enter a float value
GETDOUBLE - prompt the user to enter a double value

SYNTAX
GETFLOAT(text, f1)
GETDOUBLE(text, d1)

DESCRIPTION
The macro GETFLOAT prompts the user to enter a float value and stores this value into a float AU variable. It is used to get the value for a TOPSPIN (float) parameter which can then be stored with STOREPAR. GETFLOAT takes 2 arguments:
1. a text string which prompts the user to enter an float value
2. the float variable into which the value is store

The description for GETDOUBLE is equivalent, except that it is used for TOPSPIN (double) parameters.

EXAMPLE
The following AU program prompts the user for the frequency offset and Gaussian broadening, stores these values into the parameters O1 and GB respectively and then runs an acquisition, Gaussian multiplication and Fourier transform:

GETDOUBLE("Please enter the frequency offset:", d1)
STOREPAR("o1", d1);
GETFLOAT("Please enter the Gaussian broadening:", f1)
STOREPAR("GB", f1)
ZG
GM
FT
QUIT
SEE ALSO

GETINT - prompt the user to enter an integer value
GETSTRING - prompt the user to enter a text string
GETSTRING

NAME
GETSTRING - prompt the user to enter a text string

SYNTAX
GETSTRING(text, cval)

DESCRIPTION
The macro GETSTRING prompts the user to enter a text string which is then stored into an AU variable. It can be used for various purposes, for example to ask the user a question or prompt the user to enter a name. GETINT takes two arguments:

1. a text string which prompts the user to enter a text string
2. the character-string variable into which the value is stored

EXAMPLE
The following AU program asks the user if an integration must be done and, if yes, which intrng file must be used. Then the integrals are calculated and listed:

char answer[8];
GETSTRING("Do you want to do an integration (yes/no)?", answer)
if ( !strcmp(answer,"yes") )
{
    GETSTRING("Which intrng file must be used?", text)
    RMISC("intrng", text)
    LI
}
QUIT

SEE ALSO
GETINT - prompt the user to enter an integer value
GETFLOAT - prompt the user to enter a float value
GETDOUBLE - prompt the user to enter a double value
Chapter 12

Bruker library functions

This chapter contains a description of various C functions which are available as part of Bruker libraries. Several of them concern the handling of dataset lists or directory lists. You can, for instance, get a list of filenames, display it, select a file from the list and then copy the file to a different directory. The functions described in this chapter are particularly useful for files located in the directories /<tshome>/conf and /<tshome>/exp. For copying datasets, we recommend to use the macros described in Chapter 5.
CalcExpTime, PrintExpTime

NAME
CalcExpTime - calculate the experiment time for the current experiment
PrintExpTime - print the experiment time for the current experiment

SYNTAX
static void PrintExpTime();
int CalcExpTime();
void PrintExpTime(int exptime, int expno);
#include<inc/exptutil>

DESCRIPTION
The function CalcExpTime calculates the experiment time for the current experiment. The return value is the experiment time in seconds. The function PrintExpTime can be used to print the experiment time in the form "days hours minutes seconds".

EXAMPLE
The following AU program calculates and prints the experiment time of a sequence of 10 experiments starting with the foreground dataset.

    static void PrintExpTime();

    TIMES(10)
        PrintExpTime(CalcExpTime(),loopcount1);
    IEXPNO
    END
    QUIT
    #include<inc/exptutil>

Note that the declaration of PrintExpTime must appear at the beginning of the AU program and the #include statement at the end of the AU program.

SEE ALSO
multiexpt - a standard Bruker library AU program
CheckSumFile

NAME

CheckSumFile - creates a checksum of a data file

SYNTAX

CheckSumFile(filnam, 0, dest, 0, verb, bytord, dtyp, dim, siz0, siz, xdim)

DESCRIPTION

The function CheckSumFile generates a checksum of a data file. The output consist of a checksum preceded by hash type and data sizes, e.g:

data hash MD5: 512 * 256
16 A5 E9 14 FB 66 8B 48 09 8B E3 CA 86 D2 68 A2

which stored in a destination character string. The input data file can be a TOPSPIN raw or processed data file or any other integer data file. The data size, storage mode and dimensionality must be specified as arguments.

The arguments of the function have the following meaning:

• const char* filnam
  Full pathname of the input data file.

• char* dest
  Destination character string for function output. Must at least be 128 byte.

• int verb
  Verbose error if the input file does not exist (0=no, 1=yes)

• int bytord
  Byteorder of the input data (parameter BYTORDA for raw data or BYTORDP for processed data).

• int dtyp
  Data type of the input data (parameter DTYPA for raw data and DTYPP for processed data)

• int dim
  Data dimensionality (1 for 1D data, 2 for 2D data, ... etc.)
• int siz0
  For raw data, siz0 must be set to TD in the acquisition direction.
  For processed data, siz0 must be set to SI in the first direction.

• const int* siz
  Array of sizes. For processed data, siz must be set to SI in the successive directions. For example, for 2D data, siz = (F2-SI, F1-SI). For raw data, siz must be set to TD in the successive directions.

  Attention: in the acquisition direction, siz must be TD rounded to the next multiple of 256.

• const int* xdim
  For raw data, xdim = siz. For processed data, xdim is the array of submatrix sizes of successive directions. For example, for 2D processed data xdim = (F2-XDIM, F1-XDIM).

CheckSumFile can have the following return values:

  > 0 : successful operation
  = 0 : parameter inconsistency or I/O problems
  < 0 : all other cases

The return value can be used as an argument of the function CheckSumError() which generates an error string.

The output of CheckSumFile can be appended to the audit file with the function AuditAppend as shown in the example below.

EXAMPLE

The following AU statements will generate a data checksum of the current processed 2D data and store it in the current data auditp.txt file. It is part of the Bruker AU program xfshear.

```au
int bytordp, dtypp, size[2], xdim[2];
char name2rr[PATH_MAX], nameaudit[PATH_MAX], audittext[512];
char* auditp = audittext + sprintf(audittext, "processing description");
FETCHPARS("BYTORDP", &bytordp)
FETCHPARS("DTYP", &dtypp)
FETCHPARS("SI", &size[0])
```
FETCHPARS("XDIM", &xdim[0])
FETCHPAR1S("SI", &size[1])
FETCHPAR1S("XDIM", &xdim[1])

sprintf(name2rr, "%s/data/%s/nmr/%s/%d/pdata/%d/2rr", disk, user, name, expno, procno);
if (CheckSumFile(name2rr, 0, auditp, 0, 1, bytordp, dtypp, 2, size[0], size, xdim) > 0)
{
    sprintf(nameaudit, "%s/data/%s/nmr/%s/%d/pdata/%d/auditp.txt", disk, user, name, expno, procno);
    AuditAppend(nameaudit, audittext);
}
QUIT

SEE ALSO
AuditAppend, AuditCreate
AuditAppend

NAME

AuditAppend - appends a new entry to an existing audit file

SYNTAX

AuditAppend(const char* auditn, const char* what)

DESCRIPTION

The function AuditAppend appends a new entry to an existing audit file. An audit file entry consists of the following fields:

    NUMBER, WHEN, WHO, WHERE, PROCESS, VERSION, WHAT

All of these are automatically set by AuditAppend, except for the WHAT field which is specified as the second argument. It can be any character string.

Note that AuditAppend does not create an audit file if this does not exist yet.

EXAMPLE

See the example of the function CheckSumFile.

SEE ALSO

CheckSumFile, AuditCreate
AuditCreate

NAME

AuditCreate - create a new audit file

SYNTAX

AuditCreate(const char* auditn, const char* what)

DESCRIPTION

The function AuditCreate creates a new audit file with a single entry. This is, for example, useful when new data are created. An audit trail entry consists of the following fields:

   NUMBER, WHEN, WHO, WHERE, PROCESS, VERSION, WHAT

All of these are automatically set, except for the WHAT field which is specified as the second argument.

Caution: AuditCreate overwrites a possibly existing audit file.

EXAMPLE

Please look at the Bruker AU program split3d for an example of using AuditCreate.

SEE ALSO

CheckSumFile, AuditAppend
FileSelect

NAME

FileSelect - display a list of directory entries and allow to select entries

SYNTAX

FileSelect(sourcedir, flist, &num, type);

DESCRIPTION

The function FileSelect opens a directory, shows a list of all file and directory entries and allows you to select one or more entries. The list is stored in a 2 dimensional character-string variable which can be evaluated by subsequent AU statements. FileSelect takes four arguments:

1. the source directory
2. the variable into which the list is stored
3. the variable into which the number of selected entries is stored
4. a flag which determines whether files (0) or directories (1) are listed.

FileSelect replaces the functions uxselect and getdir which were used by TOPSPIN’s predecessor XWIN-NMR.

EXAMPLES

The following AU program will make a list of all shim files and will display this list in a selection window. If an entry is selected, then the corresponding shim file is read with the macro RSH. If no entries were found or selected, the AU program aborts.

char sourcedir[200];
int num=100;
char flist[128][128];
sprintf(sourcedir, "%s/nmr/lists/bsms", PathXWinNMRExpStan());
if (i1 = FileSelect(sourcedir, flist, &num, 0)< 0)
{
  Proc_err(DEF_ERR_OPT, "Error: No shim files selected!");
{ }
else
{
   RSH(flist[0])
}
QUIT
getdir

NAME
getdir - get all file names and/or directory names within a directory

SYNTAX
int getdir (char *directory, char ***filelist, const char *match-code);

DESCRIPTION
The function getdir opens a directory and gets all file and directory names in that directory. This list is stored in a 2 dimensional character-string variable which can be evaluated by subsequent AU statements. The list can be limited by specifying a match-code; only names matching this string are entered into the list. getdir takes three arguments:

1. the source directory
2. the variable into which the list of names is stored
3. the match-code; an arbitrary string of characters

The third argument can also be "/files" to get all files but not directories, or "/dir" to get all directories but not files.

The return value of getdir is the number of successfully matched file names and/or directory names.

getdir internally allocates memory for the list of names. Officially, you must free this memory with the Bruker library function freedir. In practice, however, you can omit freedir because all memory allocated by the AU program is automatically freed when the AU program finishes.

EXAMPLES
The following AU statements will create a list of experiment directories from an TOPSPIN dataset. All entries are returned because no match-code was specified.

```
char sourcedir[200], **listfile;
sprintf (sourcedir, "%s/data/%s/%s/%s/%s/",disk,user,type,name);
i1 = getdir (sourcedir,&listfile,NULL);
```
The following AU statements will create a list of shim files starting with the letters a to p from the bsms directory.

```c
char sourcedir[200], **listfile;
sprintf(sourcedir, "%s/nmr/lists/bsms", PathXWinNMRExpStan());
i1 = getdir (sourcedir,&listfile,"[a-p]*");
```

The following AU statement will create a list of all files but not directories from the users home directory.

```c
i1 = getdir (PathSystemHome(),&listfile,"/files");
```

The following AU statement will return a list of all directory names from the users home directory.

```c
i1 = getdir (PathSystemHome(),&listfile,"/dir");
```

**SEE ALSO**

freedir - free memory allocated by getdir
freedir

NAME

freedir - free memory allocated by getdir

SYNTAX

void freedir (char **listfile);

DESCRIPTION

The function freedir frees the memory that was allocated by a getdir function call.

EXAMPLE

See the example under the function FileSelect.

SEE ALSO

gendir - get all file names and/or directory names within a directory
fileselect - display a list from which an entry can be selected by mouse-click
**NAME**

dircp - copy a file
dircp_err - return the error message that corresponds to the error return value of a dircp function call

**SYNTAX**

dircp (char *sourcefile, char *targetfile);
char *dircp_err (int return-value);

**DESCRIPTION**

The function dircp copies the sourcefile into the targetfile. If the targetfile exists, it will be overwritten. A negative number is returned if copying was not possible. The function dircp_err will return the corresponding C error message. A return value of 0 indicates successful execution.

**EXAMPLE**

The following AU program copies the title file of the foreground dataset to the users home directory.

```c
char sourcefile[200], targetfile[200];

sprintf (targetfile, "%s/title", PathSystemHome());
if ( (i1 = dircp (PROCPATH("title"),targetfile)) < 0 )
    Proc_err (DEF_ERR_OPT, dircp_err(i1));
QUIT
```

Note that PROCPATH uses a static buffer for building the pathname, which means it cannot be used to built more then one pathname at a time, e.g. it cannot be used in both dircp arguments.
gethighest

NAME

gethighest - return the next highest unused experiment number of a dataset

SYNTAX

int gethighest (char *directory);

#include <inc/sysutil>

DESCRIPTION

The function gethighest scans a directory for all subdirectories whose name is a number and returns the next highest unused number. gethighest is typically used to scan a dataset name directory of a TOPSPIN dataset. In that case, it returns the highest unused experiment number. If, for example, the highest used experiment number is 56, the function will return the value 57. The function can also be used to return the highest unused processing number of a dataset.

EXAMPLE

The following AU program will copy the current TOPSPIN experiment into the next highest unused experiment dataset.

```c
(void) sprintf (text,"%s/data/%s/nmr/%s",disk,user,name);
i1 = gethighest (text);
WRA(i1)
QUIT
#include <inc/sysutil>
```

Note that the #include statement must be included at the end of the AU program.
**getParfileDirForRead**

**NAME**

getParfileDirForRead - determine the pathname of a list file to be read

**SYNTAX**

```c
int getParfileDirForRead (const char *filename, const char *key, char *path);
```

**DESCRIPTION**

The function `getParfileDirForRead` determines the pathname of a list file to be read. The function has three arguments:

- the filename of the list file
- the type (key) of the list file: PP_DIRS, VD_DIRS etc. (see Figure 12.1)
- the pathname of the list file

The third argument contains the result of the function. For determining this pathname, the function searches for the specified filename in all source directories that are set up for the specified list type, for the current user. The first source directory in which the file is found determines the output pathname. To view or change the list of source directories:

1. Click *Options* > *Preferences* [set]
2. Click *Miscellaneous* in the left part of the dialog box.
3. Click the *Change* button of the *entry Manage source directories....*

The functions `getParfileDirForRead` and `getParfileDirForWrite` are only implemented in *TOPSPIN 2.1* and newer. The replace the functions `getstan` and `PathXWinNMR*.

**EXAMPLE**

The following AU statements are an example of the usage of the function `getParfileDirForRead`. They are part of the AU program *proc_intrng*.

```c
char intrngfilePathname[PATH_MAX];
if (getParfileDirForRead("testrng", INTRNG_DIRS, intrngfilePathname)
```
< 0)
{
Proc_err(DEF_ERR_OPT, "testrng: %s", intrngfilePathname);
ABORT
}
RMISC("intrng", intrngfilePathname)

SEE ALSO

getParfileDirForWrite - Determines pathname of list file to be written.
<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PP_DIRS</td>
<td>pulse programs</td>
</tr>
<tr>
<td>CPD_DIRS</td>
<td>cpd programs</td>
</tr>
<tr>
<td>MAC_DIRS</td>
<td>macros</td>
</tr>
<tr>
<td>PY_DIRS</td>
<td>python programs</td>
</tr>
<tr>
<td>GP_DIRS</td>
<td>gradient programs</td>
</tr>
<tr>
<td>SHAPE_DIRS</td>
<td>shape files</td>
</tr>
<tr>
<td>SP_DIRS</td>
<td>shape lists</td>
</tr>
<tr>
<td>AU_DIRS</td>
<td>AU programs</td>
</tr>
<tr>
<td>PAR_DIRS</td>
<td>parameter sets</td>
</tr>
<tr>
<td>VD_DIRS</td>
<td>delays</td>
</tr>
<tr>
<td>VP_DIRS</td>
<td>pulses</td>
</tr>
<tr>
<td>VC_DIRS</td>
<td>loop counters</td>
</tr>
<tr>
<td>VT_DIRS</td>
<td>temperatures</td>
</tr>
<tr>
<td>VA_DIRS</td>
<td>amplitudes</td>
</tr>
<tr>
<td>F1_DIRS</td>
<td>frequency channel 1</td>
</tr>
<tr>
<td>DS_DIRS</td>
<td>data sets</td>
</tr>
<tr>
<td>SCL_DIRS</td>
<td>solvent scaling</td>
</tr>
<tr>
<td>BASE_INFO_DIRS</td>
<td>base_info files</td>
</tr>
<tr>
<td>PEAKRNG_DIRS</td>
<td>peakrng files</td>
</tr>
<tr>
<td>BASLPNTS_DIRS</td>
<td>baslpnts files</td>
</tr>
<tr>
<td>BASE_INFO_DIRS</td>
<td>base_info files</td>
</tr>
<tr>
<td>PEAKLIST_DIRS</td>
<td>peaklist files</td>
</tr>
<tr>
<td>CLEVELS_DIRS</td>
<td>clevels files</td>
</tr>
<tr>
<td>REG_DIRS</td>
<td>region files</td>
</tr>
<tr>
<td>INT2DRNG_DIRS</td>
<td>int2drng files</td>
</tr>
</tbody>
</table>

**Figure 12.1**
getParfileDirForWrite

NAME
getParfileDirForWrite - determine the pathname of a list file to be written

SYNTAX
int getParfileDirForWrite (const char *filename, const char *key, char *path);

DESCRIPTION
The function getParfileDirForWrite determines the pathname of a list file to be read. The function has three arguments:

• the filename of the list file
• the type (key) of the list file: PP_DIRS, VD_DIRS etc. (see Figure 12.1 at the description of getParfileDirForRead)
• the pathname of the list file

The third argument contains the result of the function. For determining this pathname, the function searches through all source directories that are set up for the specified list type, for the current user. The first source directory that actually exists on disk ¹, determines the output pathname. To view or change the list of source directories:

1. Click Options ’ Preferences [set ]
2. Click Miscellaneous in the left part of the dialog box.
3. Click the Change button of the entry Manage source directories....

The functions getParfileDirForRead and getParfileDirForWrite are only implemented in TOPSPIN 2.1 and newer. The replace the functions getstan and PathXWinNMR*.

EXAMPLE
The following AU statements are an example of the usage of the function

1. Usually the first specified source directory
getParfileDirForWrite. They are part of the AU program **sysgenpar**.

```c
char vdpath[PATH_MAX];
if (getParfileDirForWrite("syst1list", VD_DIRS, vdpath) < 0)
{
    Proc_err(DEF_ERR_OPT, "syst1list: %s", vdpath);
    ABORT
}
```

SEE ALSO

getParfileDirForRead - Determines pathname of list file to be read.
getstan

NAME

getstan - return the pathname to the user’s current experiment directory

SYNTAX

char *getstan (char *pathname, const char *subdir);

DESCRIPTION

The function getstan returns the pathname to the user’s current experiment directory. The returned pathname can be concatenated with a known subdirectory pathname as a part of the same getstan function call.

Please note: in TOPSPIN 2.1 and newer the functions getstan and PathXWinNMR* have become obsolete and can be replaced by the functions getParfileDirForRead and getParfileDirForWrite.

EXAMPLE

The following AU program will get the pulse program of the current AU dataset. It will then prompt the user to confirm the name of the pulse program or to enter a new name. Finally, the pulse program will be shown in a TOPSPIN window.

    char pulprog[80];
    FETCHPAR("PULPROG",pulprog)
    GETSTRING ("Enter the name of the pulse program: ",pulprog);
    (void) sprintf (text,"%s/%s",getstan (NULL,"lists/pp"),pulprog);
    showfile (text);
    QUIT

NOTE

In the above example, the function call getstan (NULL,"lists/pp") returns the pathname /<tshome>/exp/stan/nmr/lists/pp. The function call getstan(NULL,NULL) returns /<tshome>/exp/stan/nmr/.
SEE ALSO

PathXWinNMR* - a class of functions which return pathnames to certain TOPSPIN directories

getParfileDirForRead - Determines pathname of list file to be read.
getParfileDirForWrite - Determines pathname of list file to be written.
getxwinvers

NAME
getxwinvers - return the current version and patchlevel of TOPSPIN

SYNTAX
int getxwinvers (char *curversion);
#include <inc/sysutil>

DESCRIPTION
The function getxwinvers returns the version and patchlevel of the currently running TOPSPIN program into the variable curversion. This variable can then be printed out.

EXAMPLE
The following AU program prints the current version and patchlevel in the status line of TOPSPIN.

char curversion[80];
i1 = getxwinvers(curversion);
show_status (curversion);
QUIT
#include <inc/sysutil>

Note that the #include statement must be included at the end of the AU program.
**NAME**

`mkudir` - create a complete directory path

**SYNTAX**

```c
int mkudir (char *directory);
```

**DESCRIPTION**

The function `mkudir` scans the specified directory for the last `/`. Then it checks recursively for the existence of all components of the directory path and creates them if necessary. The function returns -1 if an error occurs, otherwise 0.

If the full pathname is to be created, then the directory must end with a `/` (see the example below). Possible characters behind the last slash are discarded.

**EXAMPLE**

The following AU program will create a dataset directory tree which has an experiment number one higher than the current foreground dataset.

```c
(void) sprintf (text,"%s/data/%s/nmr/%s/%d/pdata/%d/",
    disk,user,name,expno+1,procno);
if (mkudir(text) < 0)
    Proc_err (DEF_ERR_OPT, "could not create \n%s",text);
QUIT
```
PathXWinNMR

NAME

PathXWinNMR - a class of functions which return pathnames to certain TOPSPIN directories

SYNTAX

char *PathXWinNMRCConf ();
char *PathXWinNMRCurDir ();
char *PathXWinNMRDotXWinNMR ();
char *PathXWinNMRExp ();
char *PathXWinNMRPlot ();
char *PathXWinNMRProg ();

DESCRIPTION

The above functions return pathnames to certain TOPSPIN mostly subdirectories of the TOPSPIN directory <tshome>. For a standard installation, <tshome> is:

- on LINUX systems: /opt/topspin
- on Windows systems: C:\Bruker

Please note: in TOPSPIN 2.1 and newer the functions getstan and PathXWinNMR* have become obsolete and can be replaced by the functions getParfileDirForRead and getParfileDirForWrite.

For a user-defined installation, <tshome> can be any directory. The following table lists the directory pathnames returned by the above functions. For examples, please check the Bruker AU program library.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *PathXWinNMRCConf</td>
<td>returns /&lt;tshome&gt;/conf</td>
</tr>
<tr>
<td>char *PathXWinNMRCurDir</td>
<td>returns /&lt;tshome&gt;/prog/curdir</td>
</tr>
<tr>
<td>char *PathXWinNMRDotXWinNMR</td>
<td>returns $HOME/.xwinnmr-host-name</td>
</tr>
<tr>
<td>char *PathXWinNMRExp</td>
<td>returns /&lt;tshome&gt;/exp</td>
</tr>
<tr>
<td>char *PathXWinNMRPlot</td>
<td>returns /&lt;tshome&gt;/plot</td>
</tr>
</tbody>
</table>
char *PathXWinNMRProg : returns /<tshome>/prog

SEE ALSO
getParfileDirForRead - Determines pathname of list file to be read.
getParfileDirForWrite - Determines pathname of list file to be written.
**pow_next**

**NAME**

pow_next - round to the next larger power of two

**SYNTAX**

```c
int pow_next (int i1);
#include <inc/sysutil>
```

**DESCRIPTION**

The function `pow_next` takes `i1` and rounds it to the next larger integer value which is a power of two. The return value of the function is this power of two value. The function has no error handling. If `i1` is smaller than 1, then the function will return 1.

**EXAMPLE**

The following AU program will return 8192 in `i2` because this is the next larger number (compared to `i1`) which is a power of two.

```c
i1 = 7000;
i2 = pow_next(i1);
QUIT
#include <inc/sysutil>
```

Note that the `#include` statement must be included at the end of the AU program.
Proc_err

NAME
Proc_err - show an error message in a TOPSPIN dialog window

SYNTAX
int Proc_err (int flag, char *format);
int Proc_err (int flag, char *format, varargs);

DESCRIPTION
The function Proc_err can be used to construct an error message which
will be displayed in a TOPSPIN dialog window. The function takes two or
three arguments:

1. a flag which determines the type and the number (2 or 3) of but-
tons in the error window.

2. the error message to be displayed. If this argument contains %d,
%f, or %s statements, then Proc_err needs a third argument
which provides the corresponding variables.

3. variables who’s values replace the corresponding %d, %f, or %s
statements of the second argument.

The first argument (flag) can have the following values:

• DEF_ERR_OPT
  The error window has one button (OK). The AU programs holds
  until the user clicks OK.

• INFO_OPT
  The error window has one button (Seen). The AU program contin-
  ues but the error window remains on the screen until it is cleared
  by another error window or the user clicks Seen.

• QUESTION_OPT
  The error window has two buttons, OK and CANCEL. Proc_err
  returns ERR_OK (0) if the OK button is clicked and
  ERRCANCEL (-1) if the CANCEL button is clicked. The return
  value is normally used by subsequent control statements to
decide whether or not to continue the AU program.
Note that the message in the Proc_err window is constructed in the same way as the C function sprintf constructs its strings.

EXAMPLE

The following examples show several possibilities of constructing error messages for the Proc_err function call.

Example for DEF_ERR_OPT:

```c
(void) sprintf (text,"%s/data/%s/nmr/%s/%d/pdata/%d/",
    disk,user,name,expno+1,procno);
Proc_err (DEF_ERR_OPT, "could not create :\n%s",text);
```

Example for QUESTION_OPT:

```c
i1 = Proc_err(QUESTION_OPT,"Continue with the AU program ?\n\nClick OK to continue, click cancel stop");
if ( i1 == ERR_OK )
{
    /* Further AU statements */
}
if ( i1 == ERR_CANCEL )
{
    ABORT
}
```

Example for INFO_OPT:

```c
i1 = 7;
i2 = 5;
Proc_err(INFO_OPT,"%d is bigger than %d",i1,i2);
```

SEE ALSO

Show_status - show a string in the status line of TOPSPIN

All AU programs from the Bruker AU program library which use Proc_err
Show_status

NAME
Show_status - show a string in the TOPSPIN status line

SYNTAX
void Show_status (char *text);

DESCRIPTION
The function Show_status displays the specified text in the TOPSPIN status line. This function can be used as an alternative to the Proc_err function. One difference to Proc_err is that there is no window that needs to be acknowledged.

EXAMPLE
The following AU program will display the line "The AU program test has started" in the status line of TOPSPIN:

(void) strcpy(text,"The AU program test has started");
Show_status (text);
QUIT

SEE ALSO
Proc_err - show a message in a TOPSPIN dialog window
showfile

NAME
showfile - show the contents of a file in a TOPSPIN window

SYNTAX
int showfile (char *file);

DESCRIPTION
The function showfile reads the specified file and displays it in a TOPSPIN window. This display is a read-only display, so the file cannot be changed.

EXAMPLE
The following AU program will show the title file in a TOPSPIN window.

(void) sprintf (text,"%s/data/%s/nmr/%s/%d/pdata/%d/title",
disk,user,name,expno,procno);
i1 = showfile (text);
QUIT
ssleep

NAME

ssleep - pause in an AU program for a certain number of seconds

SYNTAX

int ssleep (int seconds);

DESCRIPTION

The function ssleep will cause the AU program to wait with the execution of the next statement until the specified number of seconds has elapsed.

EXAMPLE

The following AU program will wait for 3 minutes before it resumes execution.

    i1 = ssleep (180);
    EFP
    QUIT

SEE ALSO

WAIT_UNTIL - hold the AU program until the specified date and time
unlinkpr

NAME

unlinkpr - delete all processed data files (1r, 1i, 2rr, 2ii etc.) of a dataset

SYNTAX

int unlinkpr (char *directory);
#include <inc/sysutil>

DESCRIPTION

The function unlinkpr deletes all processed data files (1r, 1i, 2rr, 2ii, 2ri, 2ir, dsp, dsp.hdr, dsp_low) in the specified dataset directory. There is no error check whether the files could be deleted; the return value of the function is always 0 and can be ignored.

EXAMPLE

The following AU program will delete the processed data files of the foreground dataset.

    (void) sprintf (text,"%s/data/%s/nmr/%s/%d/pdata/%d",
        disk,user,name,expno,procno);
    i1 = unlinkpr (text);
    QUIT
    #include <inc/sysutil>

Note that the #include statement must be included at the end of the AU program.
Chapter 13

List of Bruker AU programs

13.1 Short description of all Bruker AU programs

This chapter contains a list with the names and short-descriptions of all Bruker library AU programs.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs2.water</td>
<td>Performs an F2 baseline correction on a 2D dataset left and right of the water peak.</td>
</tr>
<tr>
<td>abs2D</td>
<td>Performs a baseline correction on a 2D dataset in both dimensions.</td>
</tr>
<tr>
<td>acqu_fid_ser</td>
<td>Acquires a single FID of the current 2D experiment and replaces the old fid in the ser file.</td>
</tr>
<tr>
<td>acquist</td>
<td>Set up and start acquisitions using f1, f2, f3, vt, vc, vd, vp lists.</td>
</tr>
<tr>
<td>amplstab</td>
<td>Calculates the amplitude stability based on a peaklist file.</td>
</tr>
<tr>
<td>angle</td>
<td>Perform multiple acquisitions and ft’s. This program is particularly interesting when you want to adjust the magic angle for MAS type experiments.</td>
</tr>
<tr>
<td>asclev</td>
<td>Converts the level file in the current processed data directory to ASCII and writes it to the file.</td>
</tr>
<tr>
<td>atpplot</td>
<td>Create a plot and a pdf file within ATP. Execute when the ATP button 'Print actual spectrum' is clicked.</td>
</tr>
<tr>
<td>au_cp</td>
<td>Acquisition with adjustment of decoupling power to acquisition time.</td>
</tr>
<tr>
<td>au_getl1d</td>
<td>Acquire sweep width optimized 1D spectra.</td>
</tr>
<tr>
<td>au_getlcosy</td>
<td>Acquire sweep width optimized COSY spectra.</td>
</tr>
<tr>
<td>au_getlinv</td>
<td>Acquire sweep width optimized 2D inverse spectra.</td>
</tr>
<tr>
<td>au_getlxhco</td>
<td>Acquire sweep width optimized XH correlated spectra.</td>
</tr>
<tr>
<td>au_mult</td>
<td>AU program for C13 multiplicity analysis.</td>
</tr>
<tr>
<td>au_noediff</td>
<td>noe difference spectroscopy using different expnos.</td>
</tr>
<tr>
<td>au_noemult</td>
<td>noe difference spectroscopy with multiple irradiation points for each multiplet using different expnos.</td>
</tr>
<tr>
<td>au_water</td>
<td>Acquire water-suppression spectra for use in foreground (xau,xaua).</td>
</tr>
<tr>
<td>au_watersc</td>
<td>Acquire water-suppression spectra for use in automation, e.g., with sample changer.</td>
</tr>
<tr>
<td>au_zg</td>
<td>General AU program for data acquisition.</td>
</tr>
<tr>
<td>AU Program Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>au_zg135</td>
<td>Acquire DEPT135 type spectra.</td>
</tr>
<tr>
<td>au_zgcosy</td>
<td>Acquire COSY type spectra.</td>
</tr>
<tr>
<td>au_zgglp</td>
<td>Automatic data evaluation according to GLP standards. This AU program takes O1, SW and O2 as arguments and then works like au_zg.</td>
</tr>
<tr>
<td>au_zgmr</td>
<td>Acquisition with rotation switched off.</td>
</tr>
<tr>
<td>au_zgonly</td>
<td>General AU program for data acquisition.</td>
</tr>
<tr>
<td>au_zgsino</td>
<td>Acquisition with signal to noise break up.</td>
</tr>
<tr>
<td>au_zgte</td>
<td>Acquisition with temperature setting.</td>
</tr>
<tr>
<td>aunmp_tojdx</td>
<td>Used in LIMS automation to process data. First, AUNMP is executed, then, if specified, the command given on the command line.</td>
</tr>
<tr>
<td>autoflist</td>
<td>Automatic generation of a frequency list for the peaks in the plot region of the spectrum.</td>
</tr>
<tr>
<td>autot1</td>
<td>Automatic processing of a 2D T1/T2 experiment with subsequent T1/T2 calculation.</td>
</tr>
<tr>
<td>bsms_exam</td>
<td>Example AU program which shows how to use low level functions to read or write BSMS parameters.</td>
</tr>
<tr>
<td>bsms_getlock</td>
<td>Read current LockLevel from BSMS Unit.</td>
</tr>
<tr>
<td>butselau</td>
<td>AU program for selective experiments in bnmr.</td>
</tr>
<tr>
<td>buttonau</td>
<td>AU program for basic experiments in bnmr.</td>
</tr>
<tr>
<td>butsel90</td>
<td>AU program for calibrating selective 90 pulse bnmr.</td>
</tr>
<tr>
<td>calcphhomo</td>
<td>Calculate the phase correction for the F2 and F1 dimension of homonuclear 2D experiments.</td>
</tr>
<tr>
<td>calcphinv</td>
<td>Calculate the phase correction for the F1 dimension in HMQC/HSQC type experiments.</td>
</tr>
<tr>
<td>calcplen</td>
<td>Calculate the pulse length according to the power level.</td>
</tr>
<tr>
<td>calcpowlev</td>
<td>Calculate the power level according to the pulse length.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>calctemp</td>
<td>Calculate the temperature in the probe using the chemical shift difference between the aliphatic and OH protons.</td>
</tr>
<tr>
<td>calfun</td>
<td>Calculates an FID from an arbitrary function. This AU program is especially useful when you want to create a user defined window function for the 'uwm' command.</td>
</tr>
<tr>
<td>calibo1p1o3</td>
<td>Calibrate O1 and P1 for H2O samples.</td>
</tr>
<tr>
<td>check-vtu</td>
<td>Updates TE variable from the actual temperature and store it in a separate file (edte) in the dataset.</td>
</tr>
<tr>
<td>clev</td>
<td>Automatically calculate levels for 2D data.</td>
</tr>
<tr>
<td>clspec</td>
<td>Cleans spectra from the effects of solvent suppression. Multiple Regions can filtered or deleted from the spectrum. Entries for these regions can be deleted from peak lists and integrals. Regions are requested interactively.</td>
</tr>
<tr>
<td>coiltemp</td>
<td>Read the Shim Coil Temperature.</td>
</tr>
<tr>
<td>convbin2asc</td>
<td>Writes a 1D spectrum, with or without imaginary data points, into a file in ASCII table format. Each line in the file corresponds to one data point. The resulting file, named ascii-spec.txt, can be used to import a 1D spectrum into third party software, like Matlab.</td>
</tr>
<tr>
<td>convfidtoasc</td>
<td>AU program to convert an fid data file into an ascii table containing point number and intensity values. The output file is stored in the same directory as the fid. It can be used for calculations in spread-sheet programs or other third-party software.</td>
</tr>
<tr>
<td>convto1d</td>
<td>Converts a 2D spectrum to 1D format.</td>
</tr>
<tr>
<td>covariance</td>
<td>Processes data according to Covariance NMR</td>
</tr>
<tr>
<td>dcorr</td>
<td>Enables/disables the automatic DC-offset correction procedure of the software in case of a DRU installed.</td>
</tr>
<tr>
<td>decon_t1</td>
<td>Automatic deconvolution of a 2D T1/T2 experiment.</td>
</tr>
<tr>
<td>deptcyc</td>
<td>Creates 3 DEPT experiments from 13C experiment with CPD and then performs multiple cycles of NS scans (times 2 for DEPT90).</td>
</tr>
<tr>
<td>Program</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>depthalt</td>
<td>Halt &quot;deptcyc&quot; AU program.</td>
</tr>
<tr>
<td>diffe</td>
<td>Calculate the difference spectra between expnos.</td>
</tr>
<tr>
<td>diffp</td>
<td>Calculate the difference spectra between procnos.</td>
</tr>
<tr>
<td>dosy</td>
<td>Setup for diffusion/DOSY experiments linear gradient amplitude ramp.</td>
</tr>
<tr>
<td>elim_ints</td>
<td>Eliminates regions from the intrng file that contain the solvent and/or reference signals. The result will be an intrng file where the integral trails have a more reasonable scaling and smaller integrals are better resolved.</td>
</tr>
<tr>
<td>f1ref</td>
<td>Corrects the referencing in F1 for inverse type experiments.</td>
</tr>
<tr>
<td>fidadd</td>
<td>Add up FID's in incremented expno's.</td>
</tr>
<tr>
<td>fidtoser</td>
<td>Writes a number of fids that are stored under the same NAME and incremental EXPNOs to a ser file.</td>
</tr>
<tr>
<td>getphsum</td>
<td>Reads the total phase values from the status parameters and stores them back to the actual parameters.</td>
</tr>
<tr>
<td>gifadosy</td>
<td>Gifa starter AU program.</td>
</tr>
<tr>
<td>goalternate</td>
<td>Acquire alternated X/Y measurements. N averages are acquired alternatingly in two experiments.</td>
</tr>
<tr>
<td>graderror</td>
<td>Shows error messages generated by the gradshim gradient shimming procedure.</td>
</tr>
<tr>
<td>gontp</td>
<td>Starts an ntp test program.</td>
</tr>
<tr>
<td>gradratio</td>
<td>Calculates gradient ratios for common inverse gradient pulse programs.</td>
</tr>
<tr>
<td>gradratiogs</td>
<td>Calculates gradient ratios for common inverse gradient pulse programs.</td>
</tr>
<tr>
<td>gradshimau</td>
<td>Start gradshim gradient shimming procedure.</td>
</tr>
<tr>
<td>gsaau</td>
<td>Program to start the gradshim gradient shimming procedure.</td>
</tr>
<tr>
<td>gssel_setup</td>
<td>AU program to determine the transmitter offset for 1H selective gradient shimming using 1H as observe nucleus.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>heater</td>
<td>Switch the heater on/off.</td>
</tr>
<tr>
<td>humpcal</td>
<td>Performs the 'hump test'. Measures the width of a peak at 0.55% and 0.11% of its signal height.</td>
</tr>
<tr>
<td>hwcal</td>
<td>Calculate the width of a peak at half height.</td>
</tr>
<tr>
<td>iexpno</td>
<td>Program to change to a new experiment number.</td>
</tr>
<tr>
<td>ift3d</td>
<td>Inverse Fourier transform of 3 dimensional data.</td>
</tr>
<tr>
<td>interleave</td>
<td>Perform interleaved acquisitions</td>
</tr>
<tr>
<td>ilhalt</td>
<td>Stop an interleaved acquisition which was started with the AU program interleave.</td>
</tr>
<tr>
<td>jconv_aufx</td>
<td>Converts Jeol FX data in a loop. The data must be stored with increasing extensions like proton.1, proton.2, ... etc.</td>
</tr>
<tr>
<td>listall_au</td>
<td>Scans all AU programs and extracts the name and the short description. This information is then copied into the file listall in your home directory. This list corresponds to the list you are currently reading.</td>
</tr>
<tr>
<td>loadshimZ</td>
<td>Reads the on-axis shim values from disk and loads them to the BSMS.</td>
</tr>
<tr>
<td>lock_off</td>
<td>Switch off the lock to start data acquisition on the lock channel.</td>
</tr>
<tr>
<td>lock_on</td>
<td>Switch on the lock if it has been disabled.</td>
</tr>
<tr>
<td>loopadj</td>
<td>Parameter optimization au program which calculates the lock parameters loop filter, loop gain and loop time for optimal long-time stability after adjusting lock phase and lock gain to optimal.</td>
</tr>
<tr>
<td>make2d</td>
<td>Create a new 2D dataset from the current 1D dataset. Can be used for 2D spectroscopy and relaxation experiments. F2 parameters are copied from the 1D data, F1 parameters are set to reasonable values.</td>
</tr>
<tr>
<td>mkflist</td>
<td>Automatically generates a frequency list file.</td>
</tr>
<tr>
<td>mulabel</td>
<td>Processing AU program for determination of 13C multiplicity.</td>
</tr>
<tr>
<td>Program</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>multanal</td>
<td>Processing AU program for determination of 13C multiplicity.</td>
</tr>
<tr>
<td>multi_decon</td>
<td>Automatic deconvolution of a series of 1D spectra with AI calibration.</td>
</tr>
<tr>
<td>multi_integ</td>
<td>Automatic integration of a series of 1D spectra with AI calibration.</td>
</tr>
<tr>
<td>multi_integ2</td>
<td>Automatic integration of a series of 1D spectra with calibration of the integral values.</td>
</tr>
<tr>
<td>multi_integ3</td>
<td>Automatic integration of a series of 1D spectra with AI calibration. The output is written in a format suitable for import in excel or similar desktop publishing programs.</td>
</tr>
<tr>
<td>multi_zgvd</td>
<td>Performs multiple acquisitions on increasing expnos with delays that are read from a vdlist file. Alternatively, a fixed delay can be entered.</td>
</tr>
<tr>
<td>multi_zgvt</td>
<td>Performs multiple acquisitions on increasing expnos with temperatures that are read from a vtlist file.</td>
</tr>
<tr>
<td>multicmd</td>
<td>Performs multiple commands on increasing expnos.</td>
</tr>
<tr>
<td>multicom</td>
<td>Executes a TOPSPIN command in increasing expnos.</td>
</tr>
<tr>
<td>multicyc</td>
<td>Cycles through a series of acquisitions of increasing expnos.</td>
</tr>
<tr>
<td>multiefp</td>
<td>Performs multiple &quot;efp&quot; on increasing expnos.</td>
</tr>
<tr>
<td>multiexpt</td>
<td>Calculates experimental time for multizg.</td>
</tr>
<tr>
<td>multifp</td>
<td>Performs multiple &quot;fp&quot; on increasing expnos.</td>
</tr>
<tr>
<td>multiftapk</td>
<td>Performs multiple &quot;ft;apk&quot; on increasing expnos.</td>
</tr>
<tr>
<td>multihalt</td>
<td>Halt &quot;multicyc&quot; AU program.</td>
</tr>
<tr>
<td>multimas</td>
<td>Performs multiple MAS experiments on increasing expnos.</td>
</tr>
<tr>
<td>multipcom</td>
<td>Executes a TOPSPIN command in increasing procnos.</td>
</tr>
<tr>
<td>multiwinpro</td>
<td>Performs multiple processing on increasing expnos. The program asks for the window function and its parameters.</td>
</tr>
<tr>
<td>multixfb</td>
<td>Performs multiple &quot;xfb&quot; on increasing expnos.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>multizg</td>
<td>Performs multiple acquisitions on increasing expnos.</td>
</tr>
<tr>
<td>noediff</td>
<td>noe difference spectroscopy using different expnos.</td>
</tr>
<tr>
<td>noeflist</td>
<td>Automatic generation of a frequency list with the peaks from the current plot region for noe.</td>
</tr>
<tr>
<td>noemult</td>
<td>noe difference spectroscopy with multiple irradiation points for each multiplet using different expnos.</td>
</tr>
<tr>
<td>paropt</td>
<td>Parameter optimization au program.</td>
</tr>
<tr>
<td>parray</td>
<td>Parameter optimization au program using parameter arrays. Derived from 'paropt', but several parameters may now be changed per experiment. In addition, parameters are not changed via constant increments. Instead, the values are taken from an array.</td>
</tr>
<tr>
<td>pass2d</td>
<td>Perform a PASS experiment with 5 Pi-pulses and 16 increments (samples up to 16 spinning side bands).</td>
</tr>
<tr>
<td>pecosy</td>
<td>Program to pre-process P.E.COSY raw data before 2D-FT.</td>
</tr>
<tr>
<td>phtran</td>
<td>Transfer phase correction parameters PHC0 and PHC1 into acquisition parameters PH_ref and DE.</td>
</tr>
<tr>
<td>plintfac</td>
<td>Plot integrals with different scaling factors.</td>
</tr>
<tr>
<td>plot_sino</td>
<td>Plot spectrum, scaling depends on Signal/Noise.</td>
</tr>
<tr>
<td>plot_to_file</td>
<td>Creates a postscript file of the desired plot.</td>
</tr>
<tr>
<td>plotx</td>
<td>Plots individually scaled integral regions as separate objects.</td>
</tr>
<tr>
<td>poptau</td>
<td>Parameter optimization au program using parameter arrays. Derived from 'paropt' but several parameters can be optimized. The parameters are changed according to the parameter arrays. The AU program will be started from user interface 'popt' (parameter editor).</td>
</tr>
<tr>
<td>popthalt</td>
<td>Halt &quot;popt&quot; AU program.</td>
</tr>
<tr>
<td>proc_1H</td>
<td>Processes and plots 1D spectra. Does not perform baseline correction.</td>
</tr>
<tr>
<td>proc_1d</td>
<td>Processes and plots 1D spectra.</td>
</tr>
<tr>
<td>proc_1dapks</td>
<td>Processes and plot 1D spectra. Uses 'apks' for phase correction.</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>proc_1dconlf</td>
<td>Processes and plots 1D spectra. Plots an additional spectrum on the same plot if there are integrals in the lowfield range outside delta &gt; 11.</td>
</tr>
<tr>
<td>proc_1dconlf_pr</td>
<td>Processes and plots 1D spectra. Plots an additional spectrum on the same plot if there are integrals in the lowfield range outside the plot limits.</td>
</tr>
<tr>
<td>proc_1dgglp</td>
<td>Processing AU program with automatic data evaluation according to GLP standards. This AU program takes CY as an argument and then works like proc_1d.</td>
</tr>
<tr>
<td>proc_1dlf</td>
<td>Processes and plot 1D spectra. Plots an additional lowfield plot.</td>
</tr>
<tr>
<td>proc_1dpppti</td>
<td>Processes and plots 1D spectra. Creates a special peaklist file (frequency (Hz) and half width) and prints this on the plot.</td>
</tr>
<tr>
<td>proc_1dppti</td>
<td>Processes and plots 1D spectra. Creates a peak picking list and prints this on the plot.</td>
</tr>
<tr>
<td>proc_2d</td>
<td>Processing AU program for 2D spectra without plotting.</td>
</tr>
<tr>
<td>proc_2dhom</td>
<td>Processes and plots 2D homonuclear type spectra.</td>
</tr>
<tr>
<td>proc_2dhom_2pp</td>
<td>Processes and plots 2D homonuclear type spectra with two positive projections.</td>
</tr>
<tr>
<td>proc_2dinv</td>
<td>Processes and plots 2D inverse type spectra.</td>
</tr>
<tr>
<td>proc_2dinv_2p</td>
<td>Processes and plots 2D inverse type spectra. Plots two projections.</td>
</tr>
<tr>
<td>proc_2dphf2het</td>
<td>Determines phase correction in F2 for heteronuclear spectra.</td>
</tr>
<tr>
<td>proc_2dphf2hom</td>
<td>Determines phase correction in F2 for homonuclear spectra.</td>
</tr>
<tr>
<td>proc_2dpl</td>
<td>Processes and plots 2D type spectra.</td>
</tr>
<tr>
<td>proc_2dsym</td>
<td>Processes and symmetrizes 2D type spectra.</td>
</tr>
<tr>
<td>proc_2dt1</td>
<td>Automatic processing of one 2D T1/T2 experiment with subsequent T1/T2 calculation.</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>proc_cpd135</td>
<td>Processes and plots 13C CPD and DEPT135 spectra that were acquired with the AU program au_zg135.</td>
</tr>
<tr>
<td>proc_glp</td>
<td>Automatic GLP data evaluation.</td>
</tr>
<tr>
<td>proc_intrng</td>
<td>Processes and plots 1D spectra. Uses the predefined integral range file 'testrng' for integration.</td>
</tr>
<tr>
<td>proc_MAS</td>
<td>Processes and plot 1D MAS spectra.</td>
</tr>
<tr>
<td>proc_no</td>
<td>AU program which does no processing.</td>
</tr>
<tr>
<td>proc_noe</td>
<td>Processes and plots noediff spectra.</td>
</tr>
<tr>
<td>proc_t1</td>
<td>Semi-automatic processing of multiple 2D T1/T2 experiment with subsequent T1/T2 calculation.</td>
</tr>
<tr>
<td>proc_tecalib</td>
<td>Evaluation of previous temperature calibration experiments.</td>
</tr>
<tr>
<td>psys180f1t1</td>
<td>Processing AU program for the 180° pulse calibration tests.</td>
</tr>
<tr>
<td>psysamp1s39</td>
<td>Processing AU program for the amplitude stability tests - with shaped pulse - with 30° pulse - with 90° pulse - after gradient echo (5msec, 30 G/cm) - after gradient echo (5msec, 10 G/cm) - after gradient pulse (1msec, 10G/cm).</td>
</tr>
<tr>
<td>psysb1hom</td>
<td>Processing AU program for the B1 homogeneity test.</td>
</tr>
<tr>
<td>psysb2hom</td>
<td>Processing AU program for the B2 homogeneity test.</td>
</tr>
<tr>
<td>psycancel</td>
<td>Processing AU program for the - phase cycling cancellation test - phase cycling cancellation test after gradient pulse.</td>
</tr>
<tr>
<td>psysdante1</td>
<td>Processing AU program for the dante type turn on test.</td>
</tr>
<tr>
<td>psysdecpro1</td>
<td>Processing AU program for the decoupler profile test.</td>
</tr>
<tr>
<td>psysexpro1</td>
<td>Processing AU program for the - excitation profile (16 usec gauss shape) test - excitation profile (6 msec gauss shape) test.</td>
</tr>
<tr>
<td>sysglitch</td>
<td>Processing AU program for the glitch test.</td>
</tr>
<tr>
<td>PSYSGRRECO1</td>
<td>Processing AU program for the gradient recovery test.</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>PSYSGRZPRO</td>
<td>Processing AU program for the z-gradient profile.</td>
</tr>
<tr>
<td>PSYSMODL1</td>
<td>Processing AU program for the modulator linearity test - shaped pulse modulator linearity test.</td>
</tr>
<tr>
<td>PSA0MUTL1</td>
<td>Processing AU program for the amplitude linearity test (1dB power level steps).</td>
</tr>
<tr>
<td>PSA0PHAS1ST</td>
<td>Processing AU program for the phase stability test (&quot;13° test&quot;) - shaped pulse phase stability test (16 usec gaussian shape, &quot;13° test&quot;).</td>
</tr>
<tr>
<td>PSA0PHASF1</td>
<td>Processing AU program for the phase propagation test - phase shifting test.</td>
</tr>
<tr>
<td>PSA0PULLIN1</td>
<td>Processing AU program for the amplitude linearity test - shaped pulse amplitude linearity test (pulse length *2, power level +6).</td>
</tr>
<tr>
<td>PSA0QUADIM</td>
<td>Processing AU program for the quad image suppression test.</td>
</tr>
<tr>
<td>PSA0RGTEST</td>
<td>Processing AU program for the receiver gain test (analog and digital).</td>
</tr>
<tr>
<td>PSA0SOFTP1</td>
<td>Processing AU program for the shaped pulse comparison (rectangular, gaussian, eburp1).</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>psystestab</td>
<td>Automatic processing of a 2D Temperature stability experiment, evaluation of</td>
</tr>
<tr>
<td></td>
<td>temperature and statistic of temperature stability. Can be used to process data</td>
</tr>
<tr>
<td></td>
<td>obtained with the AU program systestab.</td>
</tr>
<tr>
<td>psysturnon</td>
<td>Processing AU program for the turn on test.</td>
</tr>
<tr>
<td>pulse</td>
<td>Program to calculate attenuation value for given pulse length or nutation</td>
</tr>
<tr>
<td></td>
<td>frequency, or vice versa.</td>
</tr>
<tr>
<td>pulsecal</td>
<td>Single scan pulse calibration via stroboscopic nutation experiment.</td>
</tr>
<tr>
<td>pulsecalib</td>
<td>Offset/Pulsecalibration H2O/D2O and Offset/Pulsecalibration 13C, 15N probehead.</td>
</tr>
<tr>
<td>qnpset</td>
<td>Define the QNP parameter according to the currently defined probehead.</td>
</tr>
<tr>
<td>quadplot</td>
<td>First plots a 2D overview spectrum and then the 4 quadrants of the 2D spectrum.</td>
</tr>
<tr>
<td>queue</td>
<td>Queue data acquisition.</td>
</tr>
<tr>
<td>queue_init</td>
<td>Initialise data acquisition with the AU program queue.</td>
</tr>
<tr>
<td>queue_erga</td>
<td>Queue data acquisition.</td>
</tr>
<tr>
<td>r23mplot</td>
<td>Read 2D slices from a 3D data set and plot them.</td>
</tr>
<tr>
<td>r23mult</td>
<td>Repeatedly reads slices from a 3D data set (3rrr) into successive experiment</td>
</tr>
<tr>
<td></td>
<td>numbers.</td>
</tr>
<tr>
<td>rampXY</td>
<td>3D gradient shimming with the BSMS RCB board.</td>
</tr>
<tr>
<td>remproc</td>
<td>Automatic conversion and processing of data sets transferred via BRUKNET,</td>
</tr>
<tr>
<td></td>
<td>LIGHTNET, NMR-LINK or TCP-LINK.</td>
</tr>
<tr>
<td>repeat</td>
<td>Repeat an acquisition with exactly the same parameters, pulse program and other</td>
</tr>
<tr>
<td></td>
<td>lists.</td>
</tr>
<tr>
<td>rescale</td>
<td>Applies intensity scaling, for direct comparison of spectra acquire with</td>
</tr>
<tr>
<td></td>
<td>different RG, NS and flip angle.</td>
</tr>
<tr>
<td>secplot</td>
<td>Generate a section plot. The overview spectrum is plotted together with a</td>
</tr>
<tr>
<td></td>
<td>vertical expansion of a smaller part of the spectrum on top of it.</td>
</tr>
<tr>
<td>set2hdecgp</td>
<td>Setup AU program for standard 3D parameter sets.</td>
</tr>
<tr>
<td>Program</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>setccnh3dgp</td>
<td>Setup AU program for standard 3D parameter sets.</td>
</tr>
<tr>
<td>setdiffparm</td>
<td>Extracts diffusion sequence parameters and stores parameters for &quot;vargrad&quot; simfit fitting (T1/T2) or DOSY processing.</td>
</tr>
<tr>
<td>seteditedgp</td>
<td>Setup AU program for standard 3D parameter sets.</td>
</tr>
<tr>
<td>sethccc3dgp</td>
<td>Setup AU program for standard 3D parameter sets.</td>
</tr>
<tr>
<td>setpar3dgp</td>
<td>Setup AU program for standard 3D parameter sets.</td>
</tr>
<tr>
<td>set_srenglist</td>
<td>Set SREGLST parameter from NUC1 and SOLVENT.</td>
</tr>
<tr>
<td>simplex</td>
<td>AU program for autoshimming. It is suitable for adjustment of strongly coupled shim groups which may be far from the optimum position.</td>
</tr>
<tr>
<td>simtoseq</td>
<td>Converts data which have been recorded in digital and qsim mode to data which appear to be acquired in qseq mode.</td>
</tr>
<tr>
<td>sinocal</td>
<td>Calculates the signal to noise ratio.</td>
</tr>
<tr>
<td>split</td>
<td>Separate data obtained with interleaved acquisition.</td>
</tr>
<tr>
<td>split2D</td>
<td>Splits a processed 2D file into single 1D spectra.</td>
</tr>
<tr>
<td>split3D</td>
<td>Separate interleaved 3D data.</td>
</tr>
<tr>
<td>splitcomb</td>
<td>Combine, shift and add 2D/3D data recorded with an interleaved single or double InPhase/AntiPhase or S3E scheme.</td>
</tr>
<tr>
<td>splitcrin</td>
<td>AU program to separate interleaved 3D data</td>
</tr>
<tr>
<td>splitcrinept</td>
<td>AU program to separate interleaved 3D data.</td>
</tr>
<tr>
<td>splitdqzq</td>
<td>Combine DQ/ZQ data.</td>
</tr>
<tr>
<td>splithb</td>
<td>Separate and combine H-bond experiments.</td>
</tr>
<tr>
<td>splithmsc</td>
<td>Separate the 1J and nJ component of the HMSC experiment.</td>
</tr>
<tr>
<td>splitinvnoe</td>
<td>Separate NOE and NONOE data obtained with a pulse program like invinoef3gpsi.</td>
</tr>
<tr>
<td>splitipap</td>
<td>Create separate InPhase and AntiPhase datasets.</td>
</tr>
<tr>
<td>splitipap2</td>
<td>Create separate InPhase and AntiPhase datasets for DSSE experiments.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>splitipap3d</td>
<td>Separate Inphase/Antiphase data from 3D datasets.</td>
</tr>
<tr>
<td>splitpfids</td>
<td>Separate decoupled/non-decoupled data obtained with a pulse program like pfidsetgpsi.</td>
</tr>
<tr>
<td>splitser</td>
<td>Splits a ser file into single fids, starting with the expno which follows the ser file.</td>
</tr>
<tr>
<td>splitxf</td>
<td>Separate and combine double half filtered data.</td>
</tr>
<tr>
<td>ssd</td>
<td>Set the 2nd and 3rd dataset from the commandline.</td>
</tr>
<tr>
<td>stack1d</td>
<td>Generates a stacked plot of 1D spectra from increasing or decreasing EXPNOs or PROCNOs.</td>
</tr>
<tr>
<td>stack2d</td>
<td>Generate a 2D stack plot.</td>
</tr>
<tr>
<td>stackp1d</td>
<td>Generates a stacked plot of 3 to 12 1D spectra from increasing or decreasing EXPNOs or PROCNOs.</td>
</tr>
<tr>
<td>stdsplit</td>
<td>Splits STD pseudo 2D datasets.</td>
</tr>
<tr>
<td>suppcal</td>
<td>Calculates the width of the Water peak at 100% and 50% of the DSS signal height. The result is referred to as the 'water suppression test'.</td>
</tr>
<tr>
<td>sys180f1t1</td>
<td>Acquisition AU program for the 180° pulse calibration test with different phases.</td>
</tr>
<tr>
<td>sys180f1t2</td>
<td>Acquisition AU program for the 180° pulse calibration test with different flip angles.</td>
</tr>
<tr>
<td>sysamp1sp9</td>
<td>Acquisition AU program for the shaped pulse amplitude stability test.</td>
</tr>
<tr>
<td>sysamp1st</td>
<td>Acquisition AU program for the amplitude stability tests - with 30° pulse - with 90° pulse.</td>
</tr>
<tr>
<td>sysb1hom</td>
<td>Acquisition AU program for the B1 homogeneity test.</td>
</tr>
<tr>
<td>sysb2hom</td>
<td>Acquisition AU program for the B2 homogeneity test.</td>
</tr>
<tr>
<td>syscancel</td>
<td>Acquisition AU program for the phase cycling cancellation test.</td>
</tr>
<tr>
<td>sysdante1</td>
<td>Acquisition AU program for the dante type turn on test.</td>
</tr>
<tr>
<td>sysdecpro1</td>
<td>Acquisition AU program for the decoupler profile test.</td>
</tr>
<tr>
<td>Program</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sysexpro1</td>
<td>Acquisition AU program for the excitation profile (16 usec gauss shape) test - excitation profile (6 msec gauss shape) test.</td>
</tr>
<tr>
<td>sysgenpar</td>
<td>Preparation AU program for all HWT test programs.</td>
</tr>
<tr>
<td>sysglitch</td>
<td>Acquisition AU program for the glitch test.</td>
</tr>
<tr>
<td>sysgrcan</td>
<td>Acquisition AU program for the phase cycling cancellation test after gradient pulse.</td>
</tr>
<tr>
<td>sysgrecho</td>
<td>Acquisition AU program for the amplitude stability test after gradient echo (5msec, 30 G/cm and 5msec, 10 G/cm).</td>
</tr>
<tr>
<td>sysgrreco1</td>
<td>Acquisition AU program for the gradient recovery test.</td>
</tr>
<tr>
<td>sysgrstab</td>
<td>Acquisition AU program for the amplitude stability test after gradient pulse (1msec, 10G/cm).</td>
</tr>
<tr>
<td>sysgrzpro</td>
<td>Acquisition AU program for the z-gradient profile.</td>
</tr>
<tr>
<td>sysmodl1</td>
<td>Acquisition AU program for the modulator linearity test.</td>
</tr>
<tr>
<td>sysmods1</td>
<td>Acquisition AU program for the shaped pulse modulator linearity test.</td>
</tr>
<tr>
<td>sysmultl1</td>
<td>Acquisition AU program for the amplitude linearity test (1dB power level steps).</td>
</tr>
<tr>
<td>sysphas1sp</td>
<td>Acquisition AU program for the shaped pulse phase stability test (16 usec gaussian shape, &quot;13 degree test&quot;).</td>
</tr>
<tr>
<td>sysphas1st</td>
<td>Acquisition AU program for the phase stability test (&quot;13 degree test&quot;).</td>
</tr>
<tr>
<td>sysphasf1</td>
<td>Acquisition AU program for the phase propagation test - phase shifting test.</td>
</tr>
<tr>
<td>syspullin1</td>
<td>Acquisition AU program for the amplitude linearity test (pulse length *2, power level +6).</td>
</tr>
<tr>
<td>sysquadim</td>
<td>Acquisition AU program for the quad image suppression test.</td>
</tr>
<tr>
<td>sysrgtest</td>
<td>Acquisition AU program for the receiver gain test (analog and digital).</td>
</tr>
<tr>
<td>AU Program</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>syssoftp1</td>
<td>Acquisition AU program for the shaped pulse comparison (rectangular, gaussian, ebup1).</td>
</tr>
<tr>
<td>syssplin1</td>
<td>Acquisition AU program for the shaped pulse amplitude linearity test (pulse length *2, power level +6).</td>
</tr>
<tr>
<td>systestab</td>
<td>AU program for a temperature stability experiment performed as pseudo 2D experiment including evaluation of temperature and statistics of temperature stability.</td>
</tr>
<tr>
<td>systurnon</td>
<td>Acquisition AU program for the turn on test.</td>
</tr>
<tr>
<td>tecalib</td>
<td>AU program to determine the temperature calibration curve.</td>
</tr>
<tr>
<td>testsuite</td>
<td>Test the general functionality of a TOPSPIN release version. Basic functionality is given if this program is completed without error messages.</td>
</tr>
<tr>
<td>tmscal</td>
<td>Performs a peak picking around the TMS signal. If the two satellites from the 29Si - 1H coupling can be detected, the resolution is OK.</td>
</tr>
<tr>
<td>tune</td>
<td>Tune a probehead.</td>
</tr>
<tr>
<td>update_layout</td>
<td>Sets the parameter 'LAYOUT' in all parameter sets.</td>
</tr>
<tr>
<td>update_aunmp</td>
<td>Sets the parameter AUNMP in all parameter sets.</td>
</tr>
<tr>
<td>vtu_airflow</td>
<td>Switch the VTU AirFlow.</td>
</tr>
<tr>
<td>vtu_exam</td>
<td>Example AU program which shows how to use low level functions to read or write VTU (BVT1000/BDTC) parameters.</td>
</tr>
<tr>
<td>vtu_heater</td>
<td>Switch the VTU Heater.</td>
</tr>
<tr>
<td>writeshimZ</td>
<td>Reads the on-axis shim values and writes a pseudo shim file.</td>
</tr>
<tr>
<td>xfshear</td>
<td>Program for shearing of 2D MQMAS spectra of odd half integer quadrupolar nuclei. Data need to be acquired in States Mode</td>
</tr>
<tr>
<td>zeroim</td>
<td>Zeroe the imaginary data of a 1D or 2D data set.</td>
</tr>
<tr>
<td>zg_2Hoffon</td>
<td>General AU program for data acquisition. The lock is switched off before the acquisition is started.</td>
</tr>
<tr>
<td>zgchkte</td>
<td>Starts acquisition with zg and monitors the temperature. The experiment is halted if the current temperature differs too much from the target temperature.</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>zg_dfs</td>
<td>Calculates shape file for double frequency sweep and subsequent data-acquisition.</td>
</tr>
<tr>
<td>2df1shift</td>
<td>Shift a 2D spectrum along the F1 axis.</td>
</tr>
<tr>
<td>2dgetref</td>
<td>Gets parameters for a 2D spectrum from the 1D reference spectra: Nucleus, Frequencies, Spectral Width, and reference plot data set names. The F2 reference is taken from the second dataset. The F1 reference is taken from the third dataset.</td>
</tr>
<tr>
<td>2dshift</td>
<td>Shift 2D time domain data left or right over NSP points.</td>
</tr>
<tr>
<td>2nde</td>
<td>Set 2nd data set to new expno and 3rd data set equal to foreground data set.</td>
</tr>
<tr>
<td>2ndn</td>
<td>Set 2nd data set to new name and 3rd data set equal to foreground data set.</td>
</tr>
</tbody>
</table>
Chapter 14

TOPSPIN parameter types

This chapter contains a list of all TOPSPIN parameters grouped by their type. The type of a parameter can be integer, float, double or character-string. Several AU macros read TOPSPIN parameters into AU variables or store the value of AU variables into TOPSPIN parameters. In both cases it is important that the type of the AU variable is the same as the parameter type.
### 14.1 Integer parameters

The following TOPSPIN parameters are of the type integer:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter</th>
<th>Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSG</td>
<td>AQORDER</td>
<td>AQSEQ</td>
<td>AQ_mod</td>
</tr>
<tr>
<td>BC_mod</td>
<td>BYTORDA</td>
<td>BYTORDP</td>
<td>DATMOD</td>
</tr>
<tr>
<td>DIGMOD</td>
<td>DIGTYP</td>
<td>DS</td>
<td>EXPNO2</td>
</tr>
<tr>
<td>EXPNO3</td>
<td>FnMODE</td>
<td>FT_mod</td>
<td>HGAIN[4]</td>
</tr>
<tr>
<td>HOLDER</td>
<td>HPMOD[8]</td>
<td>HPPRGN</td>
<td>INTBC</td>
</tr>
<tr>
<td>L[32]</td>
<td>LOCSHFT</td>
<td>LPBIN</td>
<td>MC2</td>
</tr>
<tr>
<td>ME_mod</td>
<td>NBL</td>
<td>NC</td>
<td>NCOEF</td>
</tr>
<tr>
<td>NC_proc</td>
<td>NLEV</td>
<td>NS</td>
<td>NSP</td>
</tr>
<tr>
<td>NZP</td>
<td>OVERFLW</td>
<td>PARMODE</td>
<td>PH_mod</td>
</tr>
<tr>
<td>PKNL</td>
<td>POWMOD</td>
<td>PPARMOD</td>
<td>PRGAIN</td>
</tr>
<tr>
<td>PSCAL</td>
<td>PSIGN</td>
<td>PROCNO2</td>
<td>PROCNO3</td>
</tr>
<tr>
<td>QNP</td>
<td>REVERSE</td>
<td>RO</td>
<td>RSEL[10]</td>
</tr>
<tr>
<td>SI</td>
<td>STSI</td>
<td>STSR</td>
<td>SYMM</td>
</tr>
<tr>
<td>TD</td>
<td>TD0</td>
<td>TDeff</td>
<td>TDoff</td>
</tr>
<tr>
<td>TILT</td>
<td>WBST</td>
<td>WDW</td>
<td>XDIM</td>
</tr>
<tr>
<td>XGAIN[4]</td>
<td>YMAX_a</td>
<td>YMAX_p</td>
<td>YMIN_a</td>
</tr>
<tr>
<td>YMIN_p</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
14.2 Float parameters

The following TOPSPIN parameters are of the type float:

<table>
<thead>
<tr>
<th>ABSF1</th>
<th>ABSF2</th>
<th>ABSL</th>
<th>ALPHA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSFAC</td>
<td>ASSFACI</td>
<td>ASSFACX</td>
<td>ASSWID</td>
</tr>
<tr>
<td>AZFE</td>
<td>AZFW</td>
<td>BCFW</td>
<td>CNST[64]</td>
</tr>
<tr>
<td>DC</td>
<td>DE</td>
<td>D[64]</td>
<td>FCOR</td>
</tr>
<tr>
<td>FW</td>
<td>GAMMA</td>
<td>GB</td>
<td>GPX[32]</td>
</tr>
<tr>
<td>LB</td>
<td>LEV0</td>
<td>LOCPHAS</td>
<td>MASR</td>
</tr>
<tr>
<td>MAXI</td>
<td>MI</td>
<td>NOISF1</td>
<td>NOISF2</td>
</tr>
<tr>
<td>OFFSET</td>
<td>PC</td>
<td>PCPD[10]</td>
<td>PHC0</td>
</tr>
<tr>
<td>PHC1</td>
<td>PHCOR[32]</td>
<td>PH_ref</td>
<td>PL[64]</td>
</tr>
<tr>
<td>P[64]</td>
<td>RECPH</td>
<td>RG</td>
<td>SIGF1</td>
</tr>
<tr>
<td>SIGF2</td>
<td>SINO</td>
<td>SPOAL[32]</td>
<td>SPOFFS[32]</td>
</tr>
<tr>
<td>SP[32]</td>
<td>SSB</td>
<td>S_DEV</td>
<td>TE</td>
</tr>
<tr>
<td>TE2</td>
<td>TM1</td>
<td>TM2</td>
<td>TOPLEV</td>
</tr>
<tr>
<td>V9</td>
<td>VD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
14.3 Double parameters

The following TOPSPIN parameters are of the type double:

| BF1 | BF2 | BF3 | BF4 |
| BF5 | BF6 | BF7 | BF8 |
| COROFFS | CY | F1P | F2P |
| LOCKPOW | LTIME | O1 | O2 |
| O3 | O4 | O5 | O6 |
| O7 | O8 | SF | SFO1 |
| SFO2 | SFO3 | SFO4 | SFO5 |
| SFO6 | SFO7 | SFO8 | SW |
| WBSW |

14.4 Character-string parameters

The following TOPSPIN parameters are of the type character-string:

| DSLIST[16] | DU[256] | DU2[256] | DU3[256] |
| SOLVENT[32] | SPNAM0[64] | SREGLST[40] | TI[72] |
| TYPE[16] | USER[64] | USERA1[80] | USERP1[80] |
## Index

### A
- ABORT 46
- ABS 33
- ABS1 37
- ABS2 37
- ABSD 33
- ABSD1 37
- ABSD2 37
- ABSF 33
- ABSF1 33
- ABSF2 33
- ABST1 37
- ABST2 37
- ADD 35, 63
- ADD_CURDAT_TO_PORTFOLIO 43, 114
- ADD_TO_PORTFOLIO 43, 115
- ADD2D 37
- ADDC 35
- addfid command 104
- AND 35
- APK 33, 70
- APK0 33
- APK1 33
- APKF 33
- APKS 33
- AU macro 6
- aucmd.h 16, 19
- AuditAppend 130, 131
- AUDITCOMMENTA 26
- AUTOGAIN 29
- Automatic baseline correction 41
- AUTOPHASE 29
- AUTOPLOT 42, 110
- AUTOPLOT_TO_FILE 42, 112
- AUTOPLOT_WITH_PORTFOLIO 43, 117
- AUTOPLOT_WITH_PORTFOLIO_TO_FILE 43, 118
- AUTOSHIM_OFF 29
- AUTOSHIM_ON 29
- autoshimming 30

### B
- base_info file 35
- baseline correction
  - 1D user defined 33
- baslpnts file 35
- BC 33
- BCM1 37
- BCM2 37
- brukdef.h 19
- Bruker library functions 9

### C
- CalcExpTime 126
- CalcExpTime function 60
- cc compiler 16
- C-code 16
- character string parameters 178
- CheckSumFile 127
- CLOSE_PORTFOLIO 43, 116
- column of a 2D spectrum 79, 82
- compileall command 7, 8
- compiling AU programs 7
- constants 16
- control statements 15
- CONVDTA 33
- cplbruk command 8
- CPR_exec 15, 44, 50, 51, 60
- CREATE_PORTFOLIO 43, 113

### D
- DATASET 25, 60, 62, 71
- DATASET2 25, 63
- DATASET3 25, 63
- DDATASETLIST 25
DECLARE_PORTFOLIO 43
define statements 13, 16
DEG90 29
DELPAR 28
DEXPNO 25, 61, 66
dircp 137
dircp_err function 137
DIV 35
dpa command 98
DPROCNO 25, 69
DPULPROGLIST 29
DT 35
DU 26
DVTLIST 31

E
eda command 44, 96, 98, 106, 107
edau command 7, 8, 10
edc2 command 25, 115
eddosy command 28
edit mode 8
edlock command 30
edmac command 9
edmisc command 37
edo command 106, 107
edp command 44, 96, 106
EF 33
EFP 33, 64
EJ 29
EM 33
enhanced metafile 112, 118
erropt.h 19
ERRORABORT 46
Executing AU programs 8
expinstall command 7

F
F1DISCO 40
F1PROJN 40
F1PROJP 40
F1SUM 40
F2DISCO 40
F2PROJN 40
F2PROJP 40
F2SUM 40
fcntl.h 19
FETCHDOSYPAR 28
FETCHPAR 27, 96
FETCHPAR1 27, 96
FETCHPAR1S 27
FETCHPAR3 27, 96
FETCHPAR3S 27
FETCHPARM 28, 96
FETCHPARN 27
FETCHPARNS 27
FETCHPARS 27, 98
FETCHPARS1 98
FETCHPARS3 98
FETCHT1PAR 28, 96
fidtoser AU program 86
FileSelect 132
FILT 35
first order phase correction 33
float parameters 177
FMC 33
FP 33
freedir 136
FROMJDX 44, 92
FT 33, 70
ft command 9

G
Gaussian deconvolution 37
Gaussian window multiplication 34
gcc compiler 16
GDATASETLIST 25
GDCHECK 26
GDCHECKRAW 26
GDCON 37
GENFID 34
GENSER 39
GETCURDATA 20, 25
GETCURDATA2 25
GETCURDATA3 25
GETDATASET 25, 64
getdir 134
GETDOUBLE 27, 121
GETFLOAT 27, 99, 121
gethighest 138
GETINT 27, 120
GETLCOSY 42
GETLIM 42
GETLINV 42
GETLJRES 42
GETLXHCO 42
getParfileDirForRead 139
getParfileDirForWrite 142
GETPROSOL 27
getstan 144
GETSTRING 27, 123
getxwinvers 146
GF 34
GFP 34
GLIST 25
GM 34
GO 29
GPULPROGLIST 29
GVTLIST 31

H
header files 19
Hilbert Transform 34, 39
HT 34

I
IDATASETLIST 21, 25
IEXPNO 25, 51, 60, 65, 71
IFEODATASETLIST 25
IFEOPULPROGLIST 29
IFT 34
II 29
IJ 29
ILOOPCOUNTLIST 21
include statements 13, 14, 16
integer parameters 176
intrng file 35, 123
Inverse Fourier Transform 34
    2D 39
INVSF 38
IPARSETLIST 21
IPROCNO 25, 68
IPULPROGLIST 21, 29
IVTLIST 21, 31

J
JCAMP-DX file 44, 92
JCAMP-DX format 35, 90
JCONV 44, 94
Jeol dataset 44, 94

L
lastparflag variable 11
LDCON 37
LEVCALC 38
LFILTER 30
LG 30
LGAIN 30
LI 35, 123
libcb.h 19
limits.h 19
LIPP 35
LIPPF 35
listall_au AU program 9
LO 30
LOCK 30
lock power 30
LOCK_OFF 30
LOCK_ON 30
LOCNUC 30
loop gain 30
loop statements 15
loop structures 13
loop time 30
loopcount1 variable 11
loopcount2 variable 11
LOPO 30
LS 35
LTIME 30

M
Magnitude calculation 34
MAKE_ZERO_FID 29
makeau file 16
MAS unit 31
MASE 31
MASG 32
MASH 32
MASI 31
MASR 31
Q
QSIN 34
quick reference 7
QUIT 46
QUITMSG 46

R
R12 41
R13 41
R23 41
raw data 74, 76
RDATASETLIST 25
reg file 35
REV1 38
REV2 38
REXPNO 25, 67
RGA 29
RHNP 40
RHPP 40
RMISC 35, 123
ROT 30
rotation 30
ROTOFF 30
row of 2D raw data 83, 85
row of a 2D spectrum 78, 81
RPAR 28, 106
RPROCNO 25, 70
RPULPROGLIST 20, 29
RS 36
RSC 40, 79
RSER 41, 83
RSER2D 87
RSH 30
RSR 40, 78
RV 36
RVNP 40
RVPP 40
RVTLIST 20, 31

S
SAB 34
sample.h 19
second AU dataset 63
SETCURDATA 25, 60, 61
SETDATASET 26
SETPULPROG 29
SETSH 30
SETUSER 26
Show_status 153
showfile 154
Sine window multiplication 34
SINM 34
SINO 34
SOLVENT 30
Spline baseline correction 34
splitser AU program 84
SREF 34
ssleep 155
stdio.h 17, 19
stdlib.h 17, 19
STOP 46, 47
STOPMSG 46, 47
STOREDOSYPAR 28
STOREPAR 27, 100
STOREPAR1 27, 100
STOREPAR1S 27, 104
STOREPAR3 27, 100
STOREPAR3S 27, 104
STOREPARM 28
STOREPARN 27, 102
STOREPARN 28
STOREPARNs 28
STOREPARS 27, 104
STORET1PAR 28, 100
strcpy C-function 62
string.h 19
SUB1 38
SUB1D1 38
SUB1D2 38
SUB2 38
subroutines 10, 11
Sweep_OFF 30
Sweep_ON 30
SYM 38
SYMA 38
SYMJ 38
T
TABS1 41
TABS2 41
TABS3 41
Tcl/Tk scripts 6
TE2GET 31
TE2READY 31
TE2SET 31
TEGET 31
temperature unit 31
TEPAR 31
TEREADY 31
TESET 31
TF1 41
TF1P 41
TF2 41
TF2P 41
TF3 41
TF3P 41
third AU dataset 63
TILT 38
TIMES2 19
TIMES3 19
TIMESLIST 21
TM 34
TOJDX 44, 90
TOJDX5 90
Trapezoidal baseline correction 37
Trapezoidal window multiplication 34
TRF 34
TUNE 30
TUNESX 30
U
uni.h 19
unistd.h 19
unlinkpr 156
USECURPARS 19
USELASTPARS 19
user defined
    baseline correction 33
user defined variables 10, 11
util.h 19
UWM 34
V
variable assignments 13
variable declarations 13
Varian dataset 44, 93
VCONV 44, 93
view mode 8
VIEWDATA 26, 66, 69, 71, 72
viewing AU programs 8
vorspann file 16
VT 31
VTLIST 31

**W**
WAIT_UNTIL 44, 55
WMISC 35
WPAR 28, 107
WRA 26, 74
WRP 26, 75
WRPA 26, 76
WSC 41, 82
WSER 41, 85
WSERP 41
WSH 30
WSR 81

**X**
XAU 44
XAUA 44
XAUP 44
XAUPW 44
XCMD 44, 54
XF1 38
XF1M 38
XF1P 38
XF1PS 38
XF2 38
XF2M 38
XF2P 38
XF2PS 38
XFB 38
XFBM 38
XFBP 38
XFBPS 38
XHT1 39
XHT2 39
XIF1 39
XIF2 39
XMAC 44
xmac command 9

**Z**
zero order phase correction 33
ZERT1 39
ZERT2 39
ZF 36
ZG 29
zg command 9
ZP 36